NASA Technical Paper 3187

Fault Tolerance of Artificial Neural Networks With Applications in Critical Systems

Peter W. Protzel, Daniel L. Palumbo, and Michael K. Arras

April 1992

Abstract

One of the key benefits of future hardware implementations of certain artificial neural networks (ANN's) is their apparently "built-in" fault tolerance which makes them potential candidates for critical tasks with high reliability requirements. This paper investigates the fault-tolerance characteristics of time-continuous, recurrent ANN's that can be used to solve optimization problems. The principle of operation and the performance of these networks are first illustrated by using well-known model problems like the traveling salesman problem and the assignment problem. The ANN's are then subjected to up to 13 simultaneous "stuck-at-1" or "stuck-at-0" faults for network sizes of up to 900 "neurons." The effect of these faults on the performance is demonstrated and the cause for the observed fault tolerance is discussed. An application is presented in which a network performs a critical task for a real-time distributed processing system by generating new task allocations during the reconfiguration of the system. The performance degradation of the ANN under the presence of faults is investigated by large-scale simulations, and the potential benefits of delegating a critical task to a fault-tolerant network are discussed.

1. Introduction

In spite of the fast-growing complexity and power of modern computer technology, there are a number of tasks in information processing that seem to be inherently difficult if not intractable for conventional computer systems. These are tasks like pattern recognition, nonlinear adaptive control, or autonomous navigation that are routinely mastered not only by humans but also by much "simpler" biological systems. The principles of biological information processing appear to be completely different from the way that conventional computers operate. This might explain why computers have such difficulties with tasks from the "biological domain" and vice versa. Current research in neural networks addresses these issues and seeks to explore and understand these principles of biological information processing. Recent years have seen an immense growth in those activities, which produced a variety of abstract models called artificial neural networks (ANN's) that are inspired by and loosely based on our current understanding of the operation of simple biological systems.

Although most ANN's bear little resemblance to real nervous systems and do not actually claim to be biologically plausible, they try to incorporate some of the key aspects of biological information processing. These are, for example, the capability to learn and to adapt to environmental changes, the distributed storage of information, and an architecture based on many simple computational units (model "neurons") that are interconnected and operate in parallel. Several dozen distinct types of ANN's exist that have been developed for specific purposes, but a survey or classification of these types is beyond the scope of this paper. A general introduction into so-called neural computing can be found, for example, in work by Kohonen (1988); Pao (1989); Rumelhart, McClelland, and PDP Research Group (1986); Wasserman (1989); and Zornetzer, Davis, and Lau (1990).

We are especially interested in another very intriguing characteristic of biological as well as artificial neural networks, that is, their apparently *inherent* fault tolerance. The fault tolerance of conventional systems is a carefully calculated design goal that requires some form of hardware or software redundancy which increases the complexity of the system. That is, it is always possible to build a simpler system without the redundancy, and this system has the same performance under fault-free conditions as the fault-tolerant system. In contrast, the fault tolerance of neural

networks seems to be inseparable from their functional characteristics and is neither planned nor can it be removed. This fault tolerance has been demonstrated for various ANN's, but only as a side effect and without a systematic investigation of the underlying causes. (See Anderson 1983; Sejnowski and Rosenberg 1986; Hinton and Sejnowski 1986; Hutchinson and Koch 1986.) A few studies focused more explicitly on the fault tolerance (Hinton and Shallice 1989; Belfore and Johnson 1989; Petsche and Dickinson 1990), and we will discuss their approaches and results in section 5.

In this paper we will investigate a particular ANN model that was published by Hopfield in 1984 and can be used to solve certain optimization problems. In the following discussion we will adopt the term *optimization networks* for these ANN's, a term that was coined by Tank and Hopfield (1986). The network can be implemented as an electronic circuit with nonlinear operational amplifiers representing the neurons and feedback connections between the amplifiers. The resulting complex, nonlinear dynamical system has many different stable states that represent local energy minima. If the system is properly designed, then these stable states correspond to the solutions of a target optimization problem. Thus, the system "solves" the optimization problem by converging from an initial state with partial information about the solution to a local energy minimum that corresponds to a good, if not the best, solution.

Although optimization networks were initially applied to classical problems like the traveling salesman problem, we are more interested in potential applications in real-time processing and control systems. For example, an optimization network implemented in analog hardware could perform a real-time scheduling or control task as a component of a hybrid system. If this is a critical task with high reliability requirements, then the allegedly "built-in" fault tolerance of the neural network becomes a key factor. With such applications in mind, we will investigate the fault tolerance of optimization networks and quantify the performance degradation in simulated "fault-injection" experiments. A broader goal is to gain insight into the principal character of the fault tolerance of these neural networks and to explore the underlying cause.

The following two sections of this paper contain a comprehensive introduction to optimization networks. Section 2 describes the architecture and equations that govern the dynamics of the network. The principle of how to solve an optimization problem by "mapping" it onto the network is explained in section 3 for two example problems, the assignment problem (AP) and the traveling salesman problem (TSP). Readers who are already familiar with the operation of optimization networks might want to skip these introductory sections and start with section 4 which introduces a performance measure that allows a meaningful assessment of how well the network actually solves the AP and TSP. Such a performance measure is a prerequisite for quantifying the performance degradation in the presence of simulated faults that are "injected" into the network. Section 5 presents these results for the AP and TSP that are used again as model problems and discusses the cause and effect of the observed fault tolerance. Finally, section 6 describes an application in which an optimization network is used for the real-time task allocation in a fault-tolerant, distributed processing system. The network is a critical component in this application and its fault tolerance is an essential requirement for the operation of the system. Thus, we will again illustrate how this network performs under the presence of faults and quantify the performance degradation in large-scale simulations. The concluding remarks in section 7 summarize the main results and discuss the prospects of optimization networks for different application areas.

2. Optimization Networks

In 1982, Hopfield introduced a network of interconnected model neurons that function as an associative memory with stable states corresponding to stored binary patterns. The development of this model was inspired by the observed behavior of certain physical systems that exhibit

collective phenomena, such as stable magnetic orientations, as a result of the interactions among a large number of elementary components. This associative memory model is often referred to as *Hopfield's discrete model* because it uses two-state (binary) neurons and is discrete in time as well as in state space. As an extension of this work, Hopfield (1984) proved the stability of a time-continuous model that has stable states corresponding to the discrete model and can be realized in hardware by an analog electronic circuit with operational amplifiers. This model attracted much attention, especially after Hopfield and Tank demonstrated in 1985 how it can be used to solve hard optimization problems like the TSP.

Figure 1 shows a general optimization network in the form of an electrical circuit model (Hopfield and Tank 1985) with n interconnected amplifier units (neurons) as the active circuit elements. The model allows resistive feedback from any output V_j to any input u_i with a resistor value R_{ij} or a conductance $T_{ij} = 1/R_{ij}$, respectively. The current I_i can be used to provide an external input to the network. The nonlinear, sigmoidal *transfer function* that determines the relation between an input u_i and an output V_i is given by

$$V_i = \frac{1}{2}\left[1 + \tanh\left(\frac{u_i - u_s}{u_0}\right)\right] = \frac{1}{1 + \exp\left[-4\lambda\,(u_i - u_s)\right]} \tag{1}$$

where

$$\lambda = \frac{1}{2u_0} = \left.\frac{dV_i}{du_i}\right|_{u_i = u_s}$$

The parameter λ denotes the slope of the transfer function at the inflection point $u_i = u_s$ and constitutes the maximum gain of the amplifier. This transfer function is depicted in figure 2 for a particular choice of the parameters λ and u_s. The offset u_s is sometimes explicitly used as an additional parameter (Brandt et al. 1988), but it can be incorporated into the current I_i which has also the effect of shifting the transfer function horizontally.

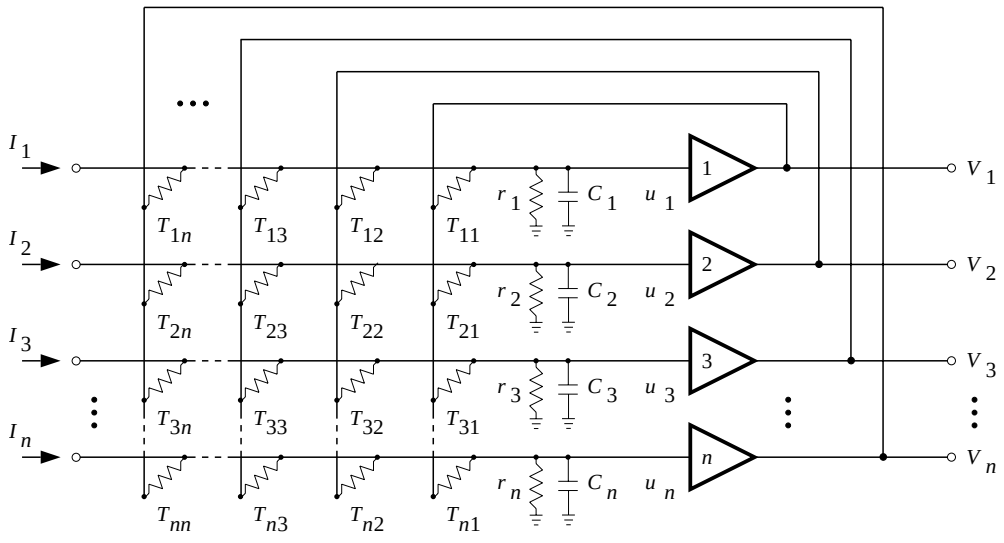


Figure 1. Circuit diagram of optimization network according to Hopfield (1984). Note that negative feedback can be realized by connecting positive conductances T_{ij} to negative output $-V_i$ of unit (not shown in this figure).

Positive and negative feedback connections, which correspond to excitatory and inhibitory synapses in biological neurons, respectively, can be mathematically described by positive and negative values for T_{ij}. Here, T_{ij} is commonly referred to as the *weight* of the connection between the output of unit j and the input of i. In an electronic circuit realization, $T_{ij} = 1/R_{ij}$ can only

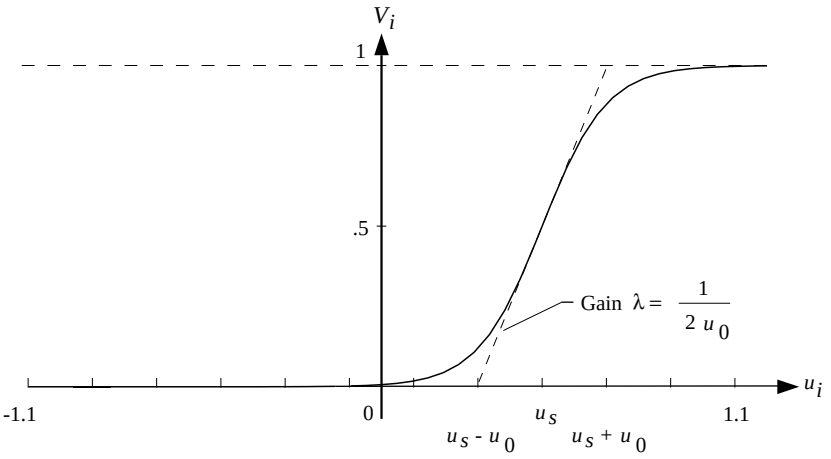


Figure 2. Nonlinear transfer function of unit. Shift $u_s = 0.5$; Gain $\lambda = 2.5$; $V_i = \frac{1}{1 + \exp[-4\lambda(u_i - u_x)]}$.

be positive, and negative feedback requires the use of an additional output $-V_i$ for unit i ranging from 0 to -1. Connecting R_{ij} to the negative output realizes negative or inhibitory feedback. The intrinsic delay exhibited by any physical amplifier as well as by a biological neuron is modeled by an input resistance r_i and capacitance C_i. These are drawn as external components in figure 1 so that the actual amplifier can be described as an ideal component with no delay.[1] A circuit analysis of the network in figure 1 yields the "equations of motion"

$$C_i \frac{du_i}{dt} = -\frac{u_i}{R_i} + \sum_{j=1}^{n} T_{ij} V_j + I_i \tag{2}$$

that describe the time evolution of the dynamical system where t denotes time. (Appendix A shows in detail how this circuit analysis is performed.) In equation (2), R_i represents the parallel combination of the input resistance r_i and all the weights $T_{ij} = 1/R_{ij}$ connected to unit i according to

$$\frac{1}{R_i} = \frac{1}{r_i} + \sum_{j=1}^{n} T_{ij} \tag{3}$$

Equation (2) is usually simplified by assuming[2] that $R_i = R$ and $C_i = C$ for all units i.

Hopfield (1984) proved the stability of the nonlinear dynamical system in equation (2) for symmetric connections ($T_{ij} = T_{ji}$). By introducing a Liapunov function, he showed that in the high-gain limit ($\lambda \rightarrow \infty$) the stable states of the system correspond to the local minima of the quantity

$$E = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} T_{ij} V_i V_j - \sum_{i=1}^{n} V_i I_i \tag{4}$$

which Hopfield refers to as the *computational energy* of the system. This means that the dynamical system moves from an initial point in state space in a direction that decreases its

[1] This is, however, an idealized model of a practical amplifier according to Smith and Portmann (1989). More realistic models might lead to instability of the system. (See also Marcus and Westervelt (1989).)

[2] Note that the assumption of a constant R_i is difficult to realize in practice because different values for the input resistances r_i would have to compensate for variations of the sum of the weights according to equation (3). These variations are considerable if problem-specific data are encoded in the weights as in the case for the TSP.

energy in equation (4) and comes to a stop at one of the many local minima of the energy function. A detailed discussion of this stability proof and the underlying assumptions can be found in appendix B.

Grossberg (1988) showed that the Liapunov function in equation (4) for the system in equation (2) is a special case of a more complex Liapunov function introduced by Cohen and Grossberg in 1983, so that equation (4) might not be considered as a new result in itself. Nevertheless, this does not diminish Hopfield and Tank's (1985) main contribution, which can be seen as their method of associating the equilibrium states of the network with the (local) solutions of an abstract optimization problem like the TSP. This method is reviewed in the next section.

3. Solving Optimization Problems: Principle of Operation

This section describes in detail how the dynamical behavior of the network can be used to solve certain optimization problems. In order to map an optimization problem onto the network, a suitable representation has to be defined and the network parameters T_{ij} and I_i have to be derived from a suitable mathematical description of the problem. Section 3.1 illustrates the basic principles by using a simple constraint satisfaction problem, which does not include a cost function but constitutes an important building block. Sections 3.2 and 3.3 then describe how a network can be used to solve two well-known optimization problems, the assignment problem (AP) and the traveling salesman problem (TSP).

3.1. Problem Representation and Constraint Satisfaction

The basic idea behind the operation of optimization networks can be stated as follows: If it is possible to associate the solutions of a particular optimization problem with the local minima of the energy function in equation (4), then the network solves the problem automatically by converging from an initial state to a local minimum, which in turn corresponds to a (local) solution of the problem. This association requires a suitable problem representation, that is, an encoding of the problem by using the state variables V_i of the network. For example, the output V_i of a unit ranging from 0 to 1 can be used to represent a certain hypothesis that is true for $V_i = 1$ and is false for $V_i = 0$. Different hypotheses can be encoded by different units and the hypotheses might have to satisfy certain constraints. If the final state of the network is supposed to represent a particular solution, it is usually required that the outputs V_i eventually converge to either 0 or 1 in order to obtain a decision. In this sense, the process of convergence with intermediate values $0 < V_i < 1$ could be interpreted as the simultaneous consideration of multiple, competing hypotheses by the network before it settles into a final state (Tagliarini and Page 1987).

A typical "building block" of optimization networks is a one-dimensional array of units that represents a set of n hypotheses under the constraint that only k out of n hypotheses can be true. Page and Tagliarini (1988) used this example to illustrate the basic principle of mapping a problem onto an optimization network. Mathematically, the problem can be stated as

$$\sum_{i=1}^{n} V_i = k \tag{5}$$

where

$$V_i \in (0,1)$$

so that exactly k out of n units are "turned on" in the final state ($k \leq n$). Note that V_i in equation (5) is a binary variable limited to the values 0 and 1. The mapping requires that

equation (5) be in the form of a quadratic function so that the minima of that function can represent the solutions to the problem. In this example, we can define the problem-specific "energy function" $E_{k,n}$ as

$$E_{k,n} = \left(\sum_{i=1}^{n} V_i - k \right)^2 + \sum_{i=1}^{n} V_i \left(1 - V_i\right) \tag{6}$$

The first term in equation (6) has minima for all combinations of V_i for which the sum of the V_i is equal to k, but this alone is not yet equivalent to equation (5) because the additional condition $V_i \in (0,1)$ has to be explicitly enforced. This is done by the second term in equation (6), which has its minima at points where V_i is either 0 or 1. After expansion of the quadratic term using the relation

$$\left(\sum_i V_i \right)^2 = \sum_i \sum_j V_i V_j$$

equation (6) can be rewritten as

$$E_{k,n} = \sum_{i=1}^{n} \sum_{j=1}^{n} V_i V_j - \sum_{i=1}^{n} V_i^2 - \sum_{i=1}^{n} V_i \left(2k - 1\right) + k^2 \tag{7}$$

The term k^2 is independent of V_i and represents only a scaling factor that can be omitted without loss of generality because the absolute value of $E_{k,n}$ is irrelevant in this context. After some further rearrangement, we get

$$E_{k,n} = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} -\ 2\left(1 - \delta_{ij}\right) V_i V_j - \sum_{i=1}^{n} V_i \left(2k - 1\right) \tag{8}$$

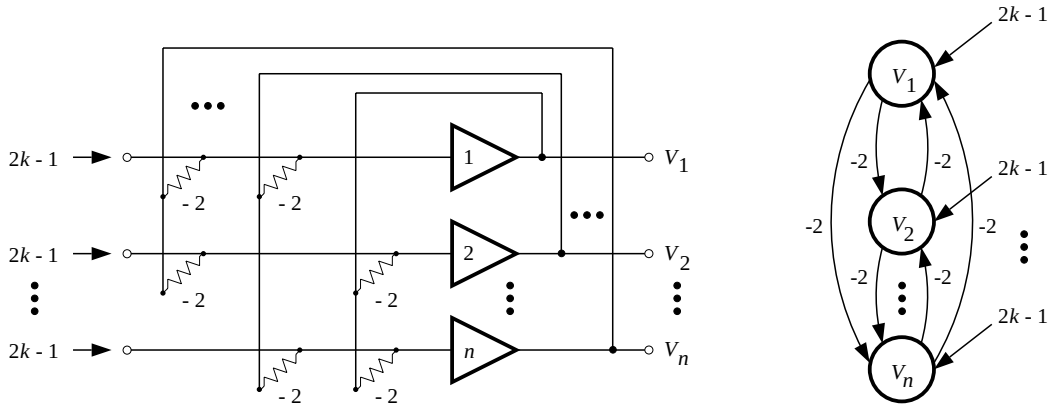
with δ_{ij} denoting the Kronecker symbol ($\delta_{ij} = 1$ for $i = j$, but 0 otherwise).

Mapping a problem onto the optimization network is equivalent to determining the network parameters T_{ij} and I_i by comparing the Liapunov function of the network (eq. (4)) with the problem-specific energy function. In our example, setting $E = E_{k,n}$ identifies the solutions of the problem (minima of $E_{k,n}$) with the stable states of the network (minima of E). With $E_{k,n}$ expressed as in equation (8), it can be seen that equations (8) and (4) are equal if $T_{ij} = -2(1 - \delta_{ij})$ and $I_i = 2k - 1$. This means that a network with n units and these parameters converges from any initial state to a final state in which k out of n outputs are *on* ($V_i = 1$) and all other outputs ($k - n$) are *off* ($V_i = 0$).[3] Figure 3(a) illustrates the resulting architecture, and figure 3(b) shows a more abstract, equivalent representation of the same network.

This kind of connectivity with negative feedback connections from every unit to every other unit is also called *lateral inhibition*. In this case, there is no negative feedback from a unit to itself, or no *self-inhibition*. Each unit i acts to inhibit all the other units with a negative feedback signal, which has a strength proportional to its current output V_i. Because all the units seem to *compete* with each other, these networks with lateral inhibition are also called *competitive networks*. Thus, the units that are on after the network reaches a stable equilibrium state are the *winners* of the competition.

Which unit converges to an on state and wins the competition depends solely on the initial values of u_i. The time evolution of the network as described by the equations of motion requires

[3] Strictly, the values $V_i = 0$ and $V_i = 1$ are reached only in the limit because of the characteristics of the sigmoidal transfer function; for practical purposes, it is sufficient to stop the simulation if $V_i > 0.95$ or $V_i < 0.05$, respectively, for all units i.



(a) Circuit level model of network with stable equilibrium states in which k out of n units are *on* ($V_i = 1$) and $k - n$ units are *off* ($V_i = 0$).

(b) A more abstract representation of same network.

Figure 3. Two architectural representations of a network.

the specification of initial values for u_i, which can be regarded as another set of inputs to the network in addition to the external currents I_i. Because of the symmetric connectivity and identical values of all weights and of all I_i, the network has an *unstable* equilibrium point ("saddle point") at $u_i = 0$ for all units i, which is equivalent to $V_i = 0.5$ for all i. Thus, an initialization with $u_i = 0$ for all i would result in no "movement" at all and would prevent the convergence of the network to any of the *stable* equilibrium points. Furthermore, an initialization with the same constant value (not necessarily 0) for all u_i would result in a movement to the unstable equilibrium point $u_i = 0$ for all i. This characteristic might be visualized by imagining the three-dimensional surface of a "saddle" with the one special curve that has a gradient pointing exactly to the (unstable) center of the saddle. (See appendix B for an illustration.)

If the initial inputs u_i do not have all the same values, then those k units with the initially largest values of u_i (and hence of V_i) suppress the other units more strongly, are less suppressed by the other units, and thus "grow even stronger" and eventually win the competition. In the n-dimensional state space spawned by the u_i, this amounts to a convergence from an initial point to the *closest* equilibrium point. These networks are also called *k-winner-take-all* networks because only the k initially strongest units converge to an on state and all other units are reduced to an off state. This characteristic can be used for *contrast enhancement* in signal processing applications (e.g., vision), and networks that use these or similar principles of competition and lateral inhibition can be found in different artificial as well as biological neural networks. For $k = 1$, the network in figure 3 is simply called a *winner-take-all* network, and the special case of $n = 2$ and $k = 1$ is equivalent to the well-known "Flip-Flop," which is a bistable memory with one unit on and the other unit off or vice versa.

The network analyzed above realizes only the satisfaction of constraints and does not include a cost function, which usually describes an optimization problem. The following sections investigate two classical examples of optimization problems, the assignment problem and the traveling salesman problem.

3.2. The Assignment Problem

The assignment problem (AP) has different variations depending on the definition of constraints and cost. The AP used for this example is a simple version, sometimes also called a *list-matching problem*, with the following specification. Given two lists of elements and a cost

value for the pairing of any two elements from these lists, the problem is to find the particular one-to-one assignment or match between the elements of the two lists that results in an overall minimum cost. In order to distinguish clearly between the two lists, we use capital letters to describe the elements of one list (i.e., $X = \mathrm{A}, \mathrm{B}, \mathrm{C}$, etc.) and enumerate the elements of the other list (i.e., $i = 1, 2, 3$, etc.). Additionally, we assume that the two lists contain the same number of elements n. A one-to-one assignment means that each element of X has to be assigned to exactly one element of i. The cost p_{Xi} for every possible assignment or pairing between X and i is given for each problem instance. This generic problem description has many practical applications, for example, the assignment of jobs i to processors X in a multiprocessor system by minimizing the cost of the communication overhead.

The AP as specified above can be represented by a two-dimensional quadratic matrix of units whose outputs are denoted by V_{Xi}. Thus, we can define V_{Xi} as a decision variable, with $V_{Xi} = 1$ meaning that the element X should be assigned to the element i, and $V_{Xi} = 0$ meaning that the pairing between X and i should not be made. This way, a solution to the AP can be uniquely encoded by the two-dimensional matrix of the outputs V_{Xi} after all units converge to 0 or 1. Note that n^2 units are required to represent an AP with n elements per list. The constraints of the one-to-one assignment require that only one unit in each row and column converge to 1 and that all other units converge to 0. Thus, the outputs of the network after convergence should produce a *permutation matrix* with exactly one unit on in each row and column. Figure 4 illustrates this representation by showing the cost matrix as the input for a particular problem instance and the output of the network after convergence. In this example, the output matrix determines the assignment of elements A to 7, B to 1, C to 6, etc.

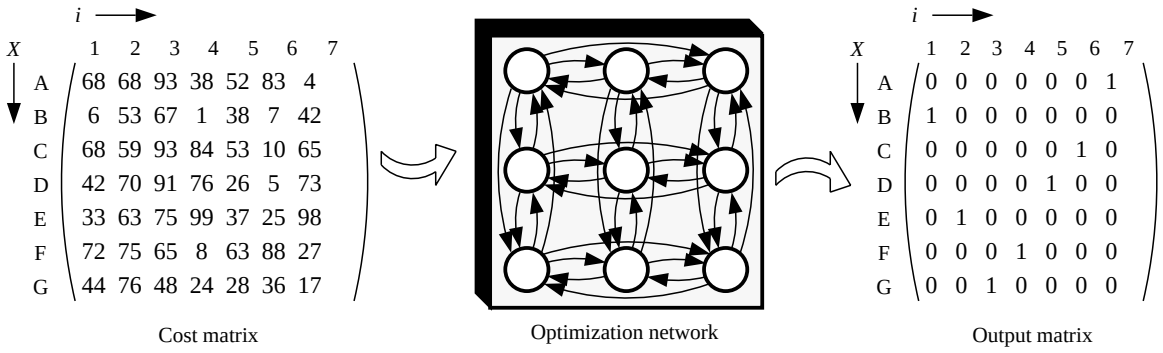


Figure 4. Exemplary cost matrix for 7×7 assignment problem and corresponding output matrix generated by neural network. Here, the solution encoded by output matrix is optimal with overall cost c of 165.

Mathematically, the constraints can be expressed as

$$\sum_X V_{Xi} = 1 \tag{9a}$$

for all units i and as

$$\sum_i V_{Xi} = 1 \tag{9b}$$

for all elements X with $V_{Xi} \in (0, 1)$. Assuming that the constraints are satisfied, the overall cost c of a particular solution becomes simply

$$c = \sum_X \sum_i p_{Xi} V_{Xi} \tag{10}$$

This summation over the whole matrix includes only the cost for the n terms for which $V_{Xi} = 1$, which represents the overall cost of the assignment. In the example of figure 4, the overall cost is $c = 165$, and it can be verified that this is actually the minimal cost of all possible solutions.

For the mapping of this problem formulation onto the optimization network, the relations in equations (9) and (10) are included in a quadratic function with minima representing the solutions of the problem. This is a generalization of the "winner-take-all" problem discussed in the last section with the augmentation that the AP requires a two-dimensional network and includes a cost function. The energy function

$$\begin{aligned} E_{\mathrm{AP}} = & \frac{A}{2} \sum_X \left(\sum_i V_{Xi} - 1 \right)^2 + \frac{B}{2} \sum_i \left(\sum_X V_{Xi} - 1 \right)^2 \\ & + \frac{C}{2} \sum_X \sum_i V_{Xi} (1 - V_{Xi}) + D \sum_X \sum_i p_{Xi} V_{Xi} \end{aligned} \tag{11}$$

used by Brandt et al. (1988) is such a quadratic function. The first two terms in equation (11) have minima if the sum over all outputs equals 1 for each row and each column, respectively. The third term has minima if all V_{Xi} are either 0 or 1, and together with the first two terms, it enforces the constraints according to equation (9). The fourth term in equation (11) is simply the overall cost of a particular solution (eq. (10)) given that the constraints are met. Furthermore, it is common to use constant factors A, B, C, and D (not to be confused with the row indices A, B, C, and D of a list as in fig. 4) as additional parameters in equation (11). These parameters have the effect of weighting the constraints and the cost function and allow a fine tuning of the performance as will be seen later.

Equation (11) creates an energy landscape in n^2-dimensional space with local minima corresponding to all possible solutions to the problem, i.e., all permutation matrices. However, unlike in the winner-take-all problem, the local minima now have different depths determined by the cost of a particular solution. The energy minimum corresponding to the smallest cost value (best solution) is called the *global minimum.*

The next step in mapping equation (11) onto an optimization network is the derivation of the values for the connections and external inputs. First, we have to extend the notation of the Liapunov function (eq. (4)) to two dimensions:

$$E = -\frac{1}{2} \sum_X \sum_i \sum_Y \sum_j T_{Xi,Yj} V_{Xi} V_{Yj} - \sum_X \sum_i V_{Xi} I_{Xi} \tag{12}$$

Now, $T_{Xi,Yj}$ and I_{Xi} can be derived by setting E in equation (12) equal to E_{AP} in equation (11). The algebraic calculations are analogous to the case of the winner-take-all problem, albeit somewhat more complex. An expansion of equation (11) results in

$$\begin{aligned} E_{\mathrm{AP}} = & \frac{A}{2} \sum_X \sum_i \sum_j V_{Xi} V_{Xj} + \frac{B}{2} \sum_X \sum_i \sum_Y V_{Xi} V_{Yi} - \frac{C}{2} \sum_X \sum_i V_{Xi}^2 \\ & - \sum_X \sum_i V_{Xi} \left(A + B - \frac{C}{2} - D p_{Xi} \right) + \frac{n}{2} (A + B) \end{aligned} \tag{13}$$

The constant scaling term $\frac{n}{2}(A+B)$ can be omitted because the absolute value of E_{AP} is not important. By using the Kronecker symbol δ_{ij} (where $\delta_{ij}=1$ for $i=j$, but 0 otherwise), we can express E_{AP} as

$$E_{\mathrm{AP}} = \sum_X \sum_i \sum_Y \sum_j V_{Xi} V_{Yj} \left(\frac{A}{2}\,\delta_{XY} + \frac{B}{2}\,\delta_{ij} - \frac{C}{2}\,\delta_{XY}\delta_{ij} \right) - \sum_X \sum_i V_{Xi} \left(A + B - \frac{C}{2} - D p_{Xi} \right) \tag{14}$$

By comparing equations (12) and (14) it can be seen that $E = E_{\mathrm{AP}}$ if

$$\left.\begin{aligned} T_{Xi,Yj} &= -A\delta_{XY} - B\delta_{ij} + C\delta_{XY}\delta_{ij} \\ I_{Xi} &= A + B - \frac{C}{2} - D p_{Xi} \end{aligned}\right\} \tag{15}$$

Figure 5 presents a sketch of the resulting network architecture. We can distinguish between three different types of connections: (1) lateral inhibitory connections between different units within the same row ($X=Y, i\neq j$) with the value $T_{Xi,Xj}=-A$, (2) lateral inhibitory connections between different units within the same column ($X\neq Y, i=j$) with the value $T_{Xi,Yi}=-B$, and (3) feedback from a unit to itself ($X=Y, i=j$) with the value $T_{Xi,Xi}=-A-B+C$. The external current includes a constant term $A+B-(C/2)$ as well as the problem-specific cost values p_{Xi}.

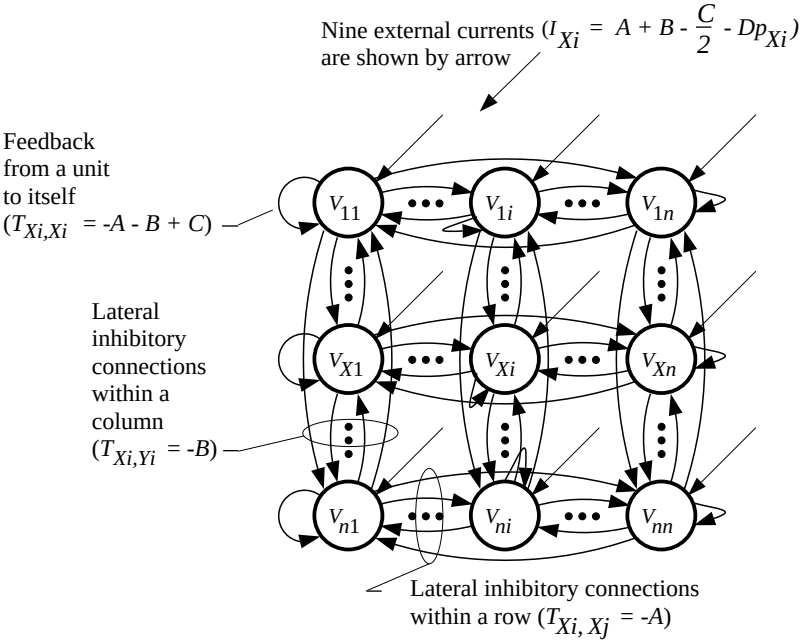


Figure 5. Schematic architecture of two-dimensional neural network with connectivity required to solve assignment problem.

The operation of the network can be simulated by solving the equations of motion (eq. (2)), which take the general form

$$C_{Xi}\,\frac{du_{Xi}}{dt} = -\frac{u_{Xi}}{R_{Xi}} + \sum_Y \sum_j T_{Xi,Yj} V_{Yj} + I_{Xi} \tag{16}$$

for a two-dimensional network. With the specific values from equation (15), the equations of motion for the AP become

$$C_{Xi}\,\frac{du_{Xi}}{dt} = -\frac{u_{Xi}}{R_{Xi}} - A\sum_{j} V_{Xj} - B\sum_{Y} V_{Yi} + CV_{Xi} + A + B - \frac{C}{2} - Dp_{Xi} \tag{17}$$

These equations represent a system of nonlinear ordinary differential equations (ODE's) that can be solved by any of the standard numerical methods. (See, e.g., Press et al. 1986.) Because the system in equation (17) proved to be numerically quite robust, the simple Euler method is sufficient as long as the stepsize Δt is small enough. The values for all the parameters used in our simulations are given in section 4 where the performance of the network is discussed.

Solving equation (17) requires the specification of initial values for all values of u_{Xi}. Unlike the winner-take-all network, the AP network in figure 5 does not have an unstable equilibrium point (saddle point) at $u_{Xi} = 0$ because the different cost values p_{Xi} encoded in the current I_{Xi} break the symmetry and the network converges from $u_{Xi} = 0$ to one of the stable states. Since we do not assume any prior knowledge of the desired solution, the initialization at $u_{Xi} = 0$ represents an unbiased choice because it does not favor any of the stable states.

Clearly, the goal in operating the AP network is the convergence from an initial state to the *global* minimum rather than to some local minimum. Unfortunately, this can be neither guaranteed nor predicted because of the complexity of the nonlinear dynamics. Each equilibrium point has a *basin of attraction* which reflects the shape of the local minimum of the energy function in the high-dimensional state space. The basins of attraction are determined by the connections, the current I_{Xi}, and the shape of the transfer function $V_i = f(u_i)$. For the winner-take-all network, all stable states have identical basins of attraction, and the final state after convergence is solely determined by the initial value. The AP network has different basins of attraction because of the different cost values p_{Xi} associated with the stable states representing a solution.

The parameters A, B, C, and D can be used to shape the basins of attractions and thus influence the convergence, but there is no theory that could prescribe specific values to achieve a desired result. Thus, suitable values for the parameters A, B, C, and D as well as for the gain and offset of the transfer function have to be found experimentally. It is relatively easy to find an optimal set of parameters for one particular problem instance. However, the same parameters might perform poorly for a different problem with a new cost function that determines a different shape of the basins of attraction. Therefore, it is necessary to find a set of parameters that performs well for a variety of problem instances. This experimental process of adjusting the parameters to optimize the performance requires a number of test cases for which the best solution is known. These questions concerning the performance assessment are discussed in section 4.

3.3. The Traveling Salesman Problem

The traveling salesman problem (TSP) was the first example chosen by Hopfield and Tank (1985) to demonstrate how a neural network could be used to solve optimization problems. The task of the traveling salesman is to visit n cities in a closed tour in such a way that the overall length of the tour is minimal. Each city can be visited only once, and the distance between any two cities is given. The TSP is a classical, NP-complete optimization problem (Garey and Johnson 1979) for which no algorithm exists that could find a (global) solution in polynomial time. Hopfield and Tank's TSP example achieved such prominence because it was one of the first

examples of a neural network solving a problem that is intractable for conventional computers. However, as we will discuss later, the TSP was meant and should be regarded as an *example* only, and it does not suggest that a general method has been discovered that solves NP-complete optimization problems.

The problem representation for the TSP is similar to the AP and requires a two-dimensional network with outputs V_{Xi}. The difference is that the first index (X) now denotes a city, and the second index (i) describes the order in which a city is visited along the tour. The representation of a problem with n cities requires a quadratic matrix of n^2 units whose outputs V_{Xi} should converge to binary values. We define $V_{Xi} = 1$ as the decision that city X should be on the ith position of the tour. Conversely, $V_{Xi} = 0$ determines that city X should not be on the ith position. With this definition, a tour can be encoded and the problem can be solved as illustrated in figure 6. First, the distances d_{XY} between any two cities X and Y have to be derived from the city locations, which are randomly distributed on a unit square in the example in figure 6. The distance matrix is then provided to the optimization network whose outputs converge to values that allow the decoding of a tour. In figure 6, for example, the output matrix determines that city C is in the first position of the tour, city F in the second position, etc., which prescribes the tour C-F-D-G-E-B-A-C.

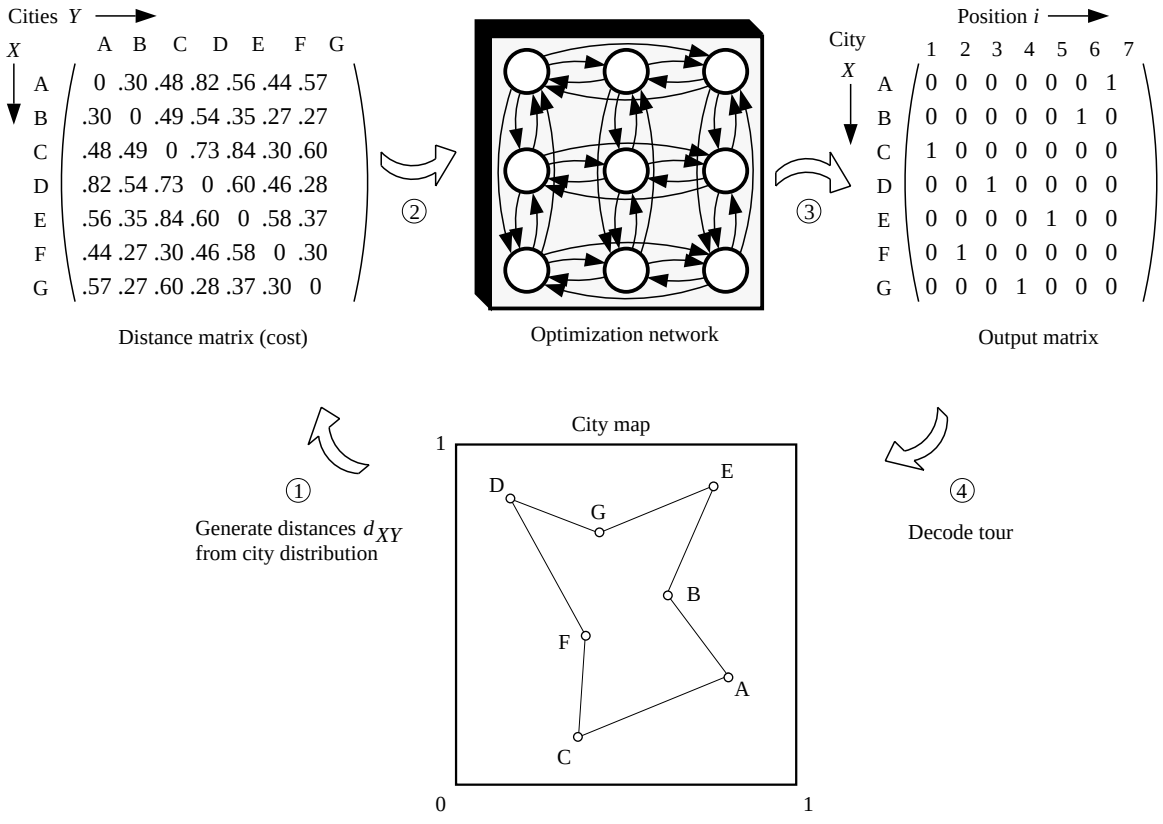


Figure 6. Example of traveling salesman problem (TSP) and representation of a tour by the outputs of the optimization network after solving the problem. The resulting tour has a length of 2.54.

Since the TSP requires a closed tour, it actually does not matter where the tour starts or in which direction the tour is traversed. Thus, the output matrix in figure 6 is not a unique description of the tour and shifting the columns to the left or to the right leads to the same result. In general, the problem representation has a $2n$-fold degeneracy because n matrices exist

for each of the two directions of traversal that encode the same tour. Although this degeneracy might seem undesirable, a more compact or unique representation is not known in this context. Furthermore, the redundancy introduced by this degeneracy has interesting implications for the fault tolerance of the network, as will be shown in section 5.

The requirement of the TSP that each city has to be visited exactly once can be rephrased such that each city can be in only one position of the tour and each position can be occupied by only one city. Thus, the constraints are met if the outputs of the network converge to a permutation matrix with only one 1 in each row and column. This means that the mathematical expression of the constraints in the form of a quadratic function is identical to the one derived for the assignment problem. The cost function for the TSP is the overall length l of a tour that should be minimized. The tour length can be expressed as (Hopfield and Tank 1985)

$$l = \frac{1}{2} \sum_i \sum_X \sum_Y d_{XY} V_{Xi} \left(V_{Y,i+1} + V_{Y,i-1} \right) \tag{18}$$

with d_{XY} denoting the distance between city X and city Y ($d_{XX} = 0$). The subscripts i describing the position are defined modulo n (i.e., $V_{Y,i+n} = V_{Y,i}$) in order to express the fact that a city in position n of the tour is adjacent to the city in position 1. Given that the constraints are met, the triple sum in equation (18) actually results in twice the overall tour length and is thus divided by 2. Equation (18) can be illustrated by the example in figure 6. Starting at position $i = 1$, the first term is $\frac{1}{2}(d_{CF} + d_{CA})$, the second term for $i = 2$ becomes $\frac{1}{2}(d_{FD} + d_{FC})$, etc. Thus, the summation includes the distances between a city in a given position and both its neighbors on the tour. The reason for including both $V_{Y,i+1}$ and $V_{Y,i-1}$ in the summation in equation (18) is that it leads to symmetric connection values in the optimization network, as we will see below. This symmetry is a necessary condition for the stability of the network. (See appendix B.)

Except for the different cost function, the energy function for the TSP is identical to that of the AP and can be written as (Brandt et al. 1988)

$$\begin{aligned} E_{\mathrm{TSP1}} = {} & \frac{A}{2} \sum_X \left(\sum_i V_{Xi} - 1 \right)^2 + \frac{B}{2} \sum_i \left(\sum_X V_{Xi} - 1 \right)^2 + \frac{C}{2} \sum_X \sum_i V_{Xi} \left(1 - V_{Xi} \right) \\ & + \frac{D}{2} \sum_X \sum_Y \sum_i d_{XY} V_{Xi} \left(V_{Y,i+1} + V_{Y,i-1} \right) \end{aligned} \tag{19}$$

The mapping of equation (19) onto the Liapunov function (eq. (12)) of the network requires calculations similar to those shown for the AP in the last section and results in the following network parameters:

$$\left. \begin{aligned} T_{Xi,Yj} &= -A\delta_{XY} - B\delta_{ij} + C\delta_{XY}\delta_{ij} - Dd_{XY} \left(\delta_{j,i+1} + \delta_{j,i-1} \right) \\ I_{Xi} &= A + B - \frac{C}{2} \end{aligned} \right\} \tag{20}$$

The principal difference between the TSP connectivity in equation (20) and the AP connectivity in equation (15) is that the TSP cost function is encoded by the connections $T_{Xi,Yj}$ and not by the external current I_{Xi}. The architecture of the TSP network is identical to the AP network as illustrated in figure 5, except that the TSP network has a constant I_{Xi} and the additional connections $T_{Xi,Yj} = -Dd_{XY}(\delta_{j,i+1} + \delta_{j,i-1})$. These connections that encode the cost function describe a link between a unit Xi and its neighbors in the two adjacent columns Y, $i+1$ and Y, $i-1$ with the strength $-Dd_{XY}$. Because of the modulo n definition of the position index,

the connections wrap around the network by connecting the first and the nth columns. A small distance between the cities X and Y, for example, is reflected by a weak inhibition between the units Xi, Y, $i+1$, and Y, $i-1$ which establish a link between X and Y in the tour if $V_{Xi} = 1$ and $V_{Y,i+1} = 1$, or $V_{Xi} = 1$ and $V_{Y,i-1} = 1$, respectively. Thus, two cities with a large distance lead to a strong inhibition between all units that could establish a link between these cities in the final tour. This competition, which favors short links to minimize the cost and leads to convergence to an overall valid tour to satisfy the constraints, occurs simultaneously in the network through the interaction of all units.

The equations of motion that describe the dynamics of the TSP network are

$$\begin{aligned} C_{Xi}\,\frac{du_{Xi}}{dt} = &-\frac{u_{Xi}}{R_{Xi}} - A\sum_{j} V_{Xj} - B\sum_{Y} V_{Yi} + C V_{Xi} \\ &- D\sum_{Y} d_{XY}\left(V_{Y,i+1} + V_{Y,i-1}\right) + A + B - \frac{C}{2} \end{aligned} \tag{21}$$

The parameters A, B, C, and D, together with the gain and the offset of the transfer function, can be used to fine tune the performance by shaping the basins of attraction as discussed in the previous section. Initial values for the u_{Xi} have to be specified in order to solve equation (21) numerically. The value $u_{Xi} = 0$ for all Xi represents an unbiased choice, but unfortunately the TSP equation (21) has an unstable equilibrium (saddle) point at $u_{Xi} = 0$. This is caused by the symmetry of the connections and, unlike the AP, by an identical external current for each unit. Unfortunately, any nonuniform initialization implies a bias toward a particular solution. Since we do not assume any prior knowledge that could be used in the form of a bias, the only solution is to keep this bias as small as possible. Thus, we use initial values $u_{Xi} + \delta$, where δ is a random variable that is uniformly distributed in the interval $-10^{-6} < \delta < +10^{-6}$. Although the random bias is fairly small, we can observe different solutions for different random initializations. This complicates the performance assessment because it requires more simulations to derive an average performance over different random initializations.

Originally, Hopfield and Tank (1985) proposed a different energy function for the TSP that used an alternative formulation to enforce the constraints. Their original TSP energy function was

$$\begin{aligned} E_{\mathrm{TSP2}} = &\frac{A}{2}\sum_{X}\sum_{i}\sum_{j\neq i} V_{Xi}V_{Xj} + \frac{B}{2}\sum_{i}\sum_{X}\sum_{Y\neq X} V_{Xi}V_{Yi} + \frac{C}{2}\left(\sum_{X}\sum_{i} V_{Xi} - n\right)^2 \\ &+ \frac{D}{2}\sum_{X}\sum_{Y}\sum_{i} d_{XY}V_{Xi}\left(V_{Y,i+1} + V_{Y,i-1}\right) \end{aligned} \tag{22}$$

The first two terms in equation (22) have a minimum (besides the trivial case $V_{Xi} = 0$ for all Xi) if all cross products $V_{Xi}V_{Xj}$ for $i \neq j$ within a row vanish and $V_{Xi}V_{Yi}$ for $X \neq Y$ within a column vanish. This is the case if there is only one nonzero output in each row and column. The third term in equation (22) has a minimum if the sum over all outputs equals n. Together with the first two terms, this determines an overall minimum if the outputs represent a permutation matrix. The mapping of equation (22) onto the Liapunov function (eq. 12)) results in the values

$$\left.\begin{aligned} T_{Xi,Yj} &= -A\delta_{XY} - B\delta_{ij} + (A+B)\,\delta_{XY}\,\delta_{ij} - C - D d_{XY}\,\left(\delta_{j,i+1} + \delta_{j,i-1}\right) \\ I_{Xi} &= nC \end{aligned}\right\} \tag{23}$$

and in the corresponding equations of motion

$$C_{Xi}\,\frac{du_{Xi}}{dt} = -\frac{u_{Xi}}{R_{Xi}} - A\sum_{j\neq i} V_{Xj} - B\sum_{Y\neq X} V_{Yi} - C\sum_{Y}\sum_{j} V_{Yj} - D\sum_{Y} d_{XY}\left(V_{Y,i+1} + V_{Y,i-1}\right) + Cn \tag{24}$$

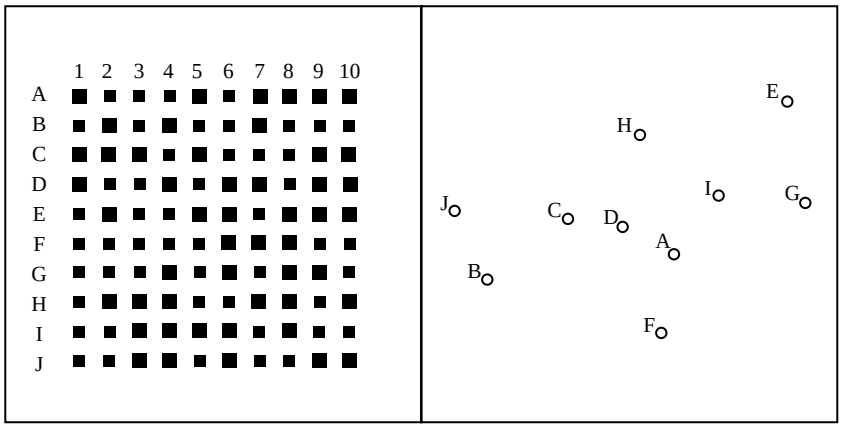
The main difference between Hopfield and Tank's original formulation (eqs. (22)–(24)) and the modification (eqs. (19)–(21)) is the *global inhibition* term $-C$ in Hopfield and Tank's equation (23) as well as an external current term that depends on the problem size n. Global inhibition means that there is an inhibitory connection from every output to every other input with a connection strength C in addition to the lateral inhibition within each row and column of strength A and B, respectively. This global connectivity results from the global formulation in equation (22), which states that the sum of all outputs should be equal to n. In contrast, the energy function in equation (19) uses only local rules when it requires that each output should converge to either 0 or 1.

Although both approaches seem to be equivalent in the sense that both enforce the convergence to a permutation matrix while using an identical cost function, their performance turns out to be considerably different. In trying to recreate Hopfield and Tank's original results, many people have reported poor results; that is, either the network failed completely to converge to a valid tour (permutation matrix) or the solution was clearly far from the global optimum. (See Wilson and Pawley 1988; Van den Bout and Miller 1988; Hedge, Sweet, and Levy 1988.) These problems do not occur when the alternative formulation of the energy function in equation (19) is used (Brandt et al. 1988). However, the performance still depends strongly on the parameter values, on the initial values, and on the cost function of the underlying city distribution.

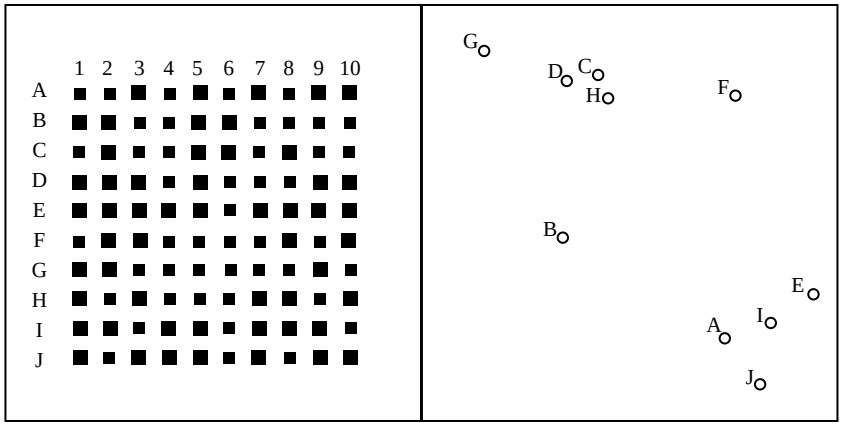
Before we address the difficulties of a quantitative performance assessment in the next section, we want to illustrate the behavior of the network in solving two 10-city distributions. For these examples, we used Brandt's equations (eqs. (19)–(21)) with the parameters $A = B = 2$, $C = 4$, $D = 1$, $\lambda = 2.5$, and $u_s = 0.5$. The equations of motion are solved by Euler's method with $\Delta t = 0.1$. The values for C_i and R_i are normalized to 1 without loss of generality. Figure 7 shows the two 10-city examples and the network in its initial state $(V_{Xi})_{t=0} = 0.5 + \delta$ with δ as a small random bias ($-10^{-6} < \delta < +10^{-6}$). The output value V_{Xi} of each neuron is represented in figure 7 by the size of the black square. This becomes more apparent in figures 8 and 9 which show the outputs of the network after convergence together with the corresponding tours.

An important point to emphasize is that the different solutions in figures 8 and 9 are caused only by different initial values and not by any other parameter variations. This illustrates the strong impact of the (unavoidable) random bias, even if it is very small. The examples also illustrate that the solutions of the network with the exception of figure 9(c) are indeed suboptimal. However, the subjective (visual) impression of a bad tour is not always reflected by a large tour length. An obviously poor solution with a twist as in figure 8(c) has a length of 2.83, which is quite close to the global optimum of 2.71. As we will discuss later, it is possible to improve the performance for specific cases by fine tuning the parameters, but this can lead to invalid answers in other cases. The parameters used here are not optimized for these examples but produce consistently valid solutions according to the results of the next section.

Finally, figure 10 shows the time evolution of a network in different snapshots during the convergence. Because of the mutual inhibition, the outputs quickly decrease from their initial values of $0.5 + \delta$ to very small values. It can be seen in figure 10(a) that the first increase in activity occurs at locations that correspond to the "city clusters" C-D-H and A-E-I-J. This result occurs because the small distances between the cities within each cluster generate less



(a) Example 1.



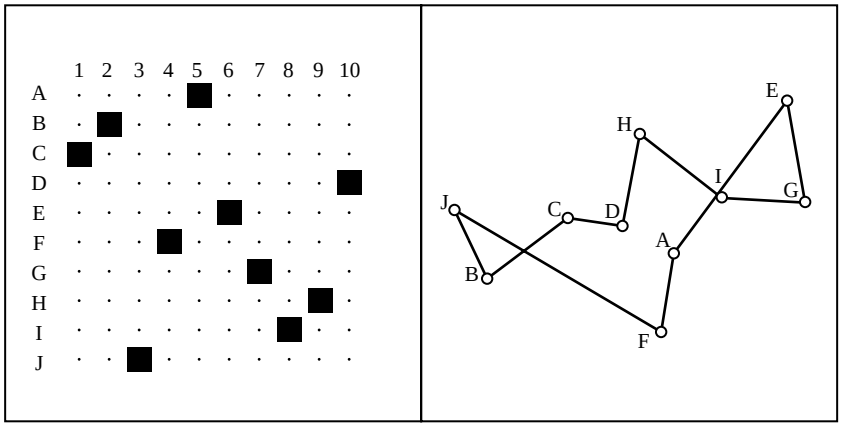
(b) Example 2.

Figure 7. Initializations of network before solving TSP and two examples of 10-city problems with cities randomly distributed on a unit square.

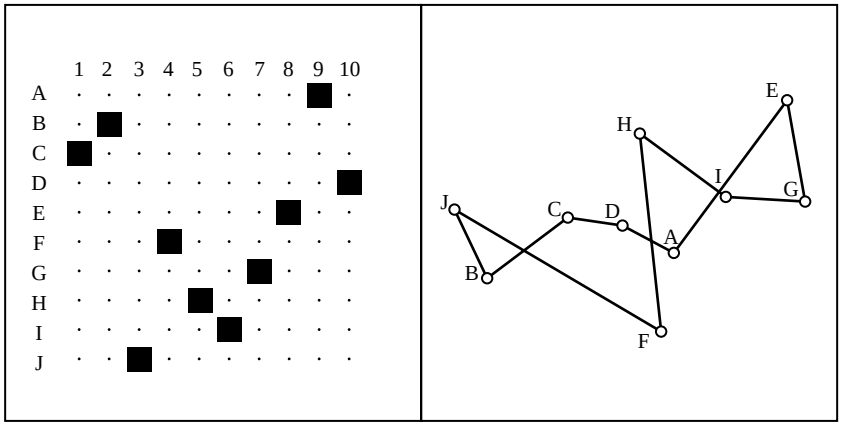
inhibition between the units at the corresponding locations. During the convergence of the network, multiple choices are considered simultaneously before the network eventually locks into a particular solution. Figure 10 gives an intuitive feeling for the meaning of the term *parallel distributed processing* that is used by some researchers as a synonym for neural computing (Rumelhart, McClelland, and PDP Research Group 1986).

4. Performance Assessment

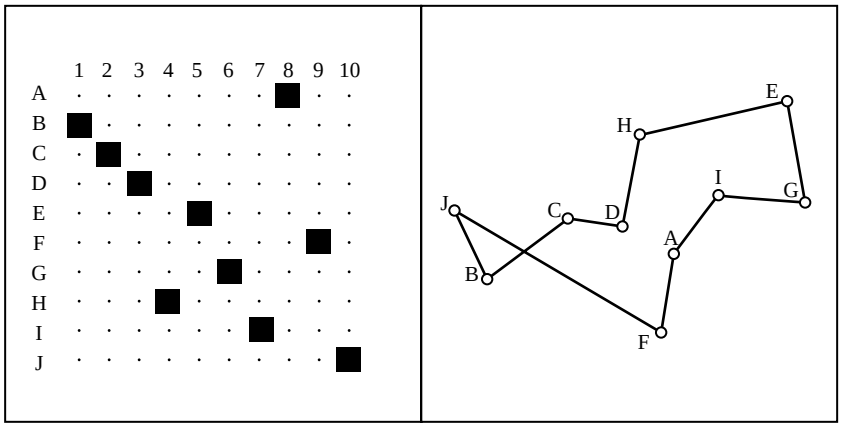
The performance assessment would not be an issue if the network simply found the global solution all the time. In fact, this would imply a solution to the NP-completeness problem. However, we have already seen that the network converges to local minima and usually produces good but suboptimal solutions. Then the question becomes *how good is good?* and the need for a performance measure arises. One obvious measure of performance is, of course, the resulting cost value after convergence, given that the network converged to a valid solution. For the TSP, this is simply the distance of the tour, and the smaller the distance the better the network performs. Unfortunately, the performance of a given network varies considerably for different problem instances (data sets), for different problem sizes, for different network parameters, and, in the case of the TSP, also for different initializations of the network. This variation



(a) Tour length, 3.01.

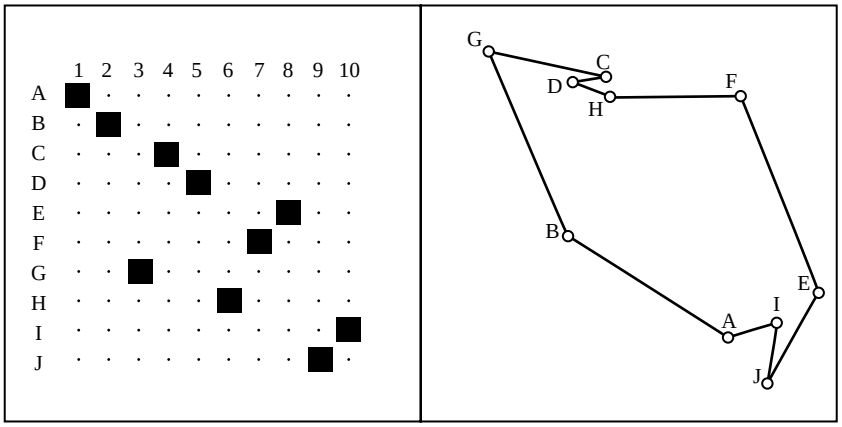


(b) Tour length, 3.23.

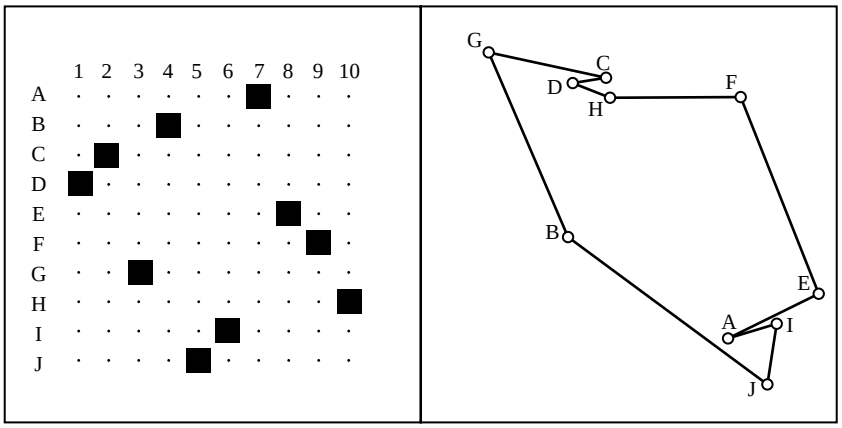


(c) Tour length, 2.83.

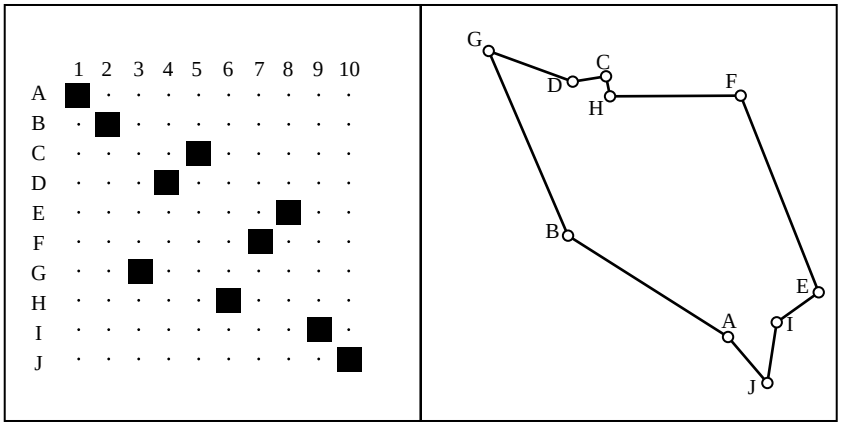
Figure 8. Different solutions of 10-city problem in figure 7(a) after different initializations of network.



(a) Tour length, 3.08.



(b) Tour length, 3.21.



(c) Tour length, 2.82.

Figure 9. Different solutions of 10-city problem in figure 7(b) after different initializations of network.

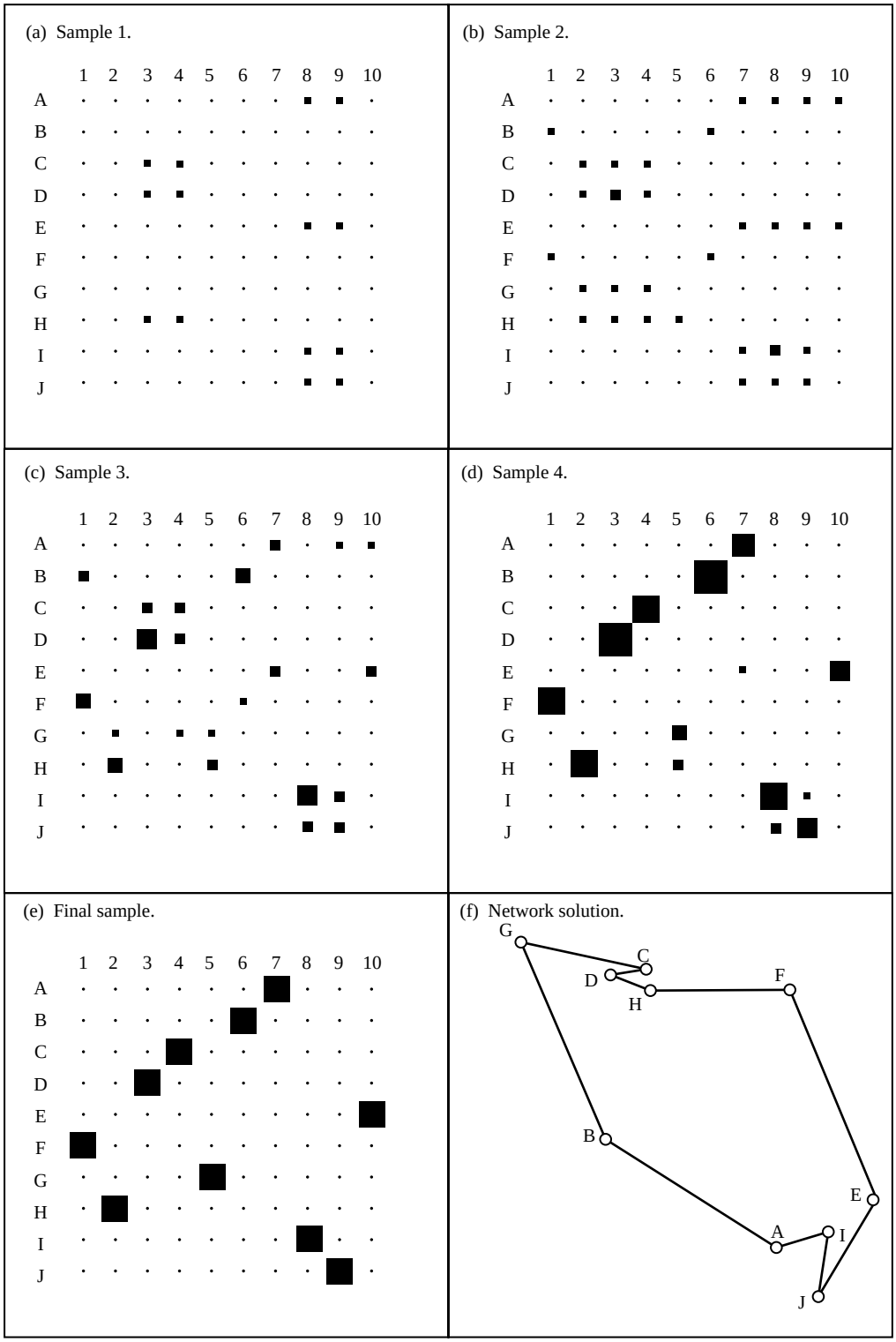


Figure 10. Time evolution of output values of network solving 10-city problem of figure 7(b). Note that solution is identical to figure 9(a) although it is encoded by a different output matrix.

impedes a meaningful, general performance assessment if only one or two example problems are considered, because it is always possible to fine tune the network parameters for a particular problem instance.

Therefore, it is necessary to generate a representative number of examples that allow a statistically meaningful statement to be made about the *average* performance. Furthermore, some reference frame is needed for the comparison of the network results because just the average over the cost values is generally not sufficient. For example, the average of the tour lengths of 100 different TSP city distributions is not significant unless the problem size is constant and the statistical distribution of the input data (city coordinates) is known. The simplest reference for a comparison is the average cost value of a random guess, that is, the average or expected value of the distribution of all possible answers for a particular problem instance. A performance assessment based on the estimated distribution has led to statements in the literature that, for example, a solution is approximately among the 10^8 best out of 4.4×10^{30} possible solutions (Hopfield and Tank 1985), or that 92 percent of the solutions are among the best 0.01 percent of all solutions (Tagliarini and Page 1987). Although this gives some impression of the performance, it can hardly be considered a practical measurement.

The solution needed is a performance measure that can answer the following questions:

1. What is the effect of a parameter variation or a modification of the energy function on the performance?

2. How good is the solution with respect to the global optimum or (the best known answer)?

3. How does the performance change with problem size?

4. With respect to fault tolerance, how does the performance degrade under the presence of (simulated) faults?

5. What is the performance difference of two networks solving two different problems; that is, are there problems that are "easier" for the network to solve?

Our approach to the performance assessment is based on the fact that the distribution of all possible answers for every instance of an optimization problem can be characterized by two values, the global optimum (minimum cost) c_{opt} and the average cost value c_{av}. With c denoting the cost value of a given result derived by the network, the relation between c, c_{opt}, and c_{av} can be used as a performance measure. By mapping those absolute values onto a normalized scale as illustrated in figure 11, we define the *solution quality* q as

$$q = \frac{c_{\text{av}} - c}{c_{\text{av}} - c_{\text{opt}}} \tag{25}$$

Thus, the solution quality has a value $q = 1$ if $c = c_{\text{opt}}$ and $q = 0$ if $c = c_{\text{av}}$, with $0 < q < 1$ for $c_{\text{av}} > c > c_{\text{opt}}$.

Obviously, the calculation of q requires the knowledge of the two reference values c_{opt} and c_{av} for each problem instance (e.g., for each city distribution of the TSP). Obtaining values for c_{av} is usually no problem since it requires only a sufficient number of random trials. In case of the TSP, for example, a random but valid tour is generated repeatedly and the resulting tour lengths are averaged to obtain c_{av}. The fact that we have to know the global optimum c_{opt} appears to be a paradox at first glance, and one might ask why we would use an ANN to solve a problem for which the best possible solution is already known. The answer is, of course, that we want to *test* the network by using well-known *model problems*, and for such a test it is reasonable to compare the results of a new method (i.e., ANN's) with the results of the best existing method. In fact, in almost all cases, where ANN's have been applied to optimization problems, there are conventional algorithms readily available to provide values for c_{opt}. For NP-complete optimization problems like the TSP, for which the global optimum is generally unknown, the best available heuristic method like the Lin-Kernighan algorithm can be used as a reference. (See Lin and Kernighan 1973.) If c_{opt} is not the global optimum and if the

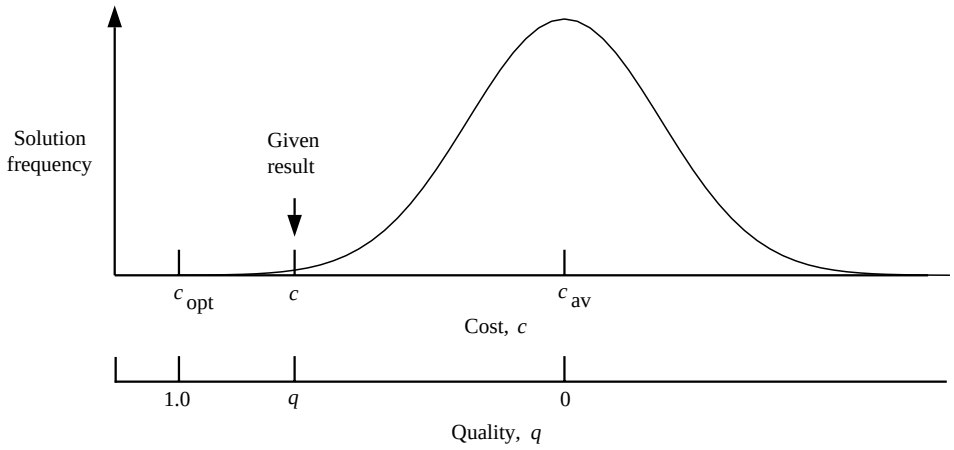


Figure 11. Definition of solution quality q by mapping absolute values of c, $c_{\rm opt}$, and $c_{\rm av}$ onto normalized scale.

network happens to generate a better answer, then the event $c < c_{\rm opt}$ is reflected by a solution quality $q > 1$. Conversely, the value for q becomes negative if the solution of the network is worse than the random average ($c > c_{\rm av}$). Thus, the normalized solution quality is independent of a particular problem instance and of the problem size.

In the following discussion we will demonstrate the use of the defined solution quality to assess and compare the performance of the two model problems, the TSP and the AP. In order to get statistically relevant results for the TSP, we generated a *test set* containing 10 different city distributions for each problem size ($n = 10$, 20, and 30) and 5 different distributions for $n = 50$ and 100. Each city distribution was generated by placing the cities randomly on a unit square according to a uniform probability distribution. The values for $c_{\rm av}$ were obtained by averaging over 10^5 random trials for each city distribution. The Lin-Kernighan (1973) algorithm was used to generate five answers for each city distribution, and the best result was chosen as $c_{\rm opt}$. After obtaining the values for $c_{\rm av}$ and $c_{\rm opt}$ for each city distribution, it is possible to calculate the solution quality q according to equation (25) after each simulation run of the network. Since the network performance varies considerably for different random initializations, 10 different initializations were used for each city distribution of $n = 10$ to 50, and 5 initializations for $n = 100$. Thus, a single sweep through the test set requires 375 simulation runs, and the value of q was calculated after each run. The average values of q are shown in table 1 for different approaches and problem sizes.

The possibility also exists that the network will not converge at all to a valid solution because it has gotten stuck in a local minimum (spurious attractor) that does not correspond to a permutation matrix. Since this event is not reflected by the solution quality, we also show in table 1 the proportion of runs with valid solutions. The average value for q includes only runs that produced valid solutions. In an attempt to recreate Hopfield and Tank's (1985) original results, we performed a run of the test set using their original equations (eqs. (22)–(24)) with the parameters $A = B = 500$, $C = 200$, $D = 500$, $\lambda = 25$, and $u_s = 0$ as described. Furthermore, Hopfield and Tank used an additional constant term for the external current according to $I_{Xi} = C(n + 5) = 200n + 1000$, which effectively shifts the transfer function. They also used the initialization $(V_{Xi})_{t=0} = (1/n) + \delta$, where δ is a small random number.

Table 1. TSP Values of Solution Quality (q) for Different Approaches and Problem Sizes

[Proportions of valid solutions are given in parentheses]

Different approaches using TSP	Values of q for problem sizes (number of cities) of—				
	$n = 10$	$n = 20$	$n = 30$	$n = 50$	$n = 100$
Original method of Hopfield and Tank (1985)	0.905 (0.15)	0.903 (0.11)	0.851 (0.02)	(0)	
Modified method of Brandt et al. (1988)	0.829 (1.00)	0.816 (1.00)	0.830 (1.00)	0.852 (1.00)	0.902 (1.00)
Different parameters of Brandt et al. (1988)	0.936 (0.98)	0.926 (0.97)	0.923 (0.84)	0.913 (0.58)	0.927 (0.18)

The equations of motion (eq. (24)) were solved by Euler's method with time steps Δt between 10^{-5} and 10^{-6}. A larger Δt can cause numerical errors and results that do not reflect the actual behavior of the system. The first row in table 1 shows the results of our simulation that confirm the reported difficulties (Wilson and Pawley 1988; Hedge, Sweet, and Levy 1988; Brandt et al. 1988) in using Hopfield and Tank's original equations. Even for $n = 10$ cities, only 15 percent of the runs converged to a valid solution, and since none of the 50-city cases produced a valid answer, we did not even attempt to solve a 100-city problem.

Although we experimented extensively with parameter variations, we did not find a set of parameters that improves the performance significantly. However, it is possible to fine tune the parameters for *one particular* city distribution to obtain quite impressive results. Unfortunately, the same parameters usually produce invalid or poor results for other city distributions. This characteristic has led to some confusion in the literature with performance claims based on specific examples that were difficult to reproduce and were not valid in general. (See Wilson and Pawley 1988.) This also demonstrates the importance of an *average* performance assessment over many examples. Since Hopfield and Tank's original equations (eqs. (22)–(24)) are not the only way to express the problem, we tried different modifications (Protzel, Palumbo, and Arras 1989; Protzel 1990) and obtained the best results with the approach published by Brandt et al. (1988) that is described in section 3.3. By using Brandt's energy equation (19) and his original parameters $A = B = 2$, $C = 4$, $D = 1$, $\lambda = 2.5$, and $u_s = 0.5$, we obtained the results shown in the second row of table 1. An additional difference of Brandt's approach is an initialization in the center of the hypercube with $(V_{Xi})_{t=0} = 0.5 + \delta$ and a random variable δ uniformly distributed in the range $-10^{-6} \leq \delta \leq 10^{-6}$. Because of the lower gain and smaller values of the parameters, we could use the value $\Delta t = 0.1$ to solve the equations of motion (eq. (21)).

As shown in the second row of table 1, this modified energy function produced consistently valid tours across the full range of problem sizes. However, the average solution quality was lower than the valid cases of Hopfield and Tank's results. We tried different parameters for Brandt's energy equations to improve the quality. The results for $A = B = 5$, $C = 2$, and $D = 3$ are listed in the third row of table 1. The parameters for the transfer function and the initialization are the same as in the previous case, except that we used $\Delta t = 5 \times 10^{-3}$. We can see that the average quality has indeed been improved, but at the price of occasional invalid answers whose frequency increases with the problem size. There is a fundamental trade-off between obtaining consistently valid (but sometimes poor) answers for a large number of different problem instances and very good answers for a small number of instances. One obvious and extreme case of this trade-off is setting $D = 0$, which cancels the cost function and reduces the problem to pure constraint satisfaction. Then, we would always expect valid answers, but with an average quality of $q = 0$. The underlying problem with the TSP is the quadratic cost function that maps the

problem-specific distance values multiplied by the parameter D onto the connections, where they are added to the values that enforce the constraints as in equations (20) or (23). Qualitatively speaking, large distance values in an extreme problem case or a large factor D might override the connectivity values that enforce the constraints and thus interfere with the convergence to a valid solution.

This problem does not occur with the assignment problem because the energy function for the AP (Hopfield 1982) maps the problem-dependent cost values to the external current (eq. (15)) and not to the connection values. This is actually the only difference between the AP and the TSP networks, as far as the architecture is concerned, and it makes a performance comparison between the problems especially interesting. As before, we generated a test set of 10 problem instances for each size of 10, 20, 30, 50, and 100 elements. The cost values were randomly generated with a uniform distribution between 0 and 1. The AP as defined here is not an NP-complete problem, and relatively simple and fast algorithms exist that find the global solution. We used such a textbook algorithm (Syslo, Deo, and Kowalik 1983) to obtain values for c_{opt} and generated the average values c_{av} from 10^5 random solutions for each problem instance. The first row of table 2 shows the simulation results for the parameters originally used by Brandt et al. (1988) with the additional values $\lambda = 2.5$, $u_s = 0.5$, $\Delta t = 0.05$, and the initialization $(u_{Xi})_{t=0} = 0$. The other two rows show the effect of parameter modifications, and here the values $\lambda = 25$, $u_s = 0$, and $\Delta t = 5 \times 10^{-5}$ were used with the same initialization. As discussed in section 3.2, no random bias in the initial values is required for the AP; in fact, the network converges to the same solution despite some small random noise. This simplifies the performance assessment considerably because we now need only one simulation run for each problem instance.

A comparison between tables 1 and 2 reveals a striking difference between the TSP and the AP results. For the AP, none of the runs failed to converge to a valid tour, and moreover the solution quality is excellent. For the parameter sets 2 and 3 in table 2, the network actually found the global optimum in most cases or generated an answer extremely close to it. We can conclude that the encoding of the cost values by the external current is the cause for the enormous performance improvement because, unlike with the TSP, the cost values do not interfere with the connection values that enforce the constraints. Thus, the distinction between a quadratic and a linear cost function becomes an important classification that helps to identify problems that are more suitable to an ANN implementation. The demonstrated ability to compare the results of two different optimization problems proves the versatility of the solution quality as a performance index and justifies the additional effort needed to obtain values for c_{opt} and c_{av}.

Table 2. AP Values of Solution Quality (q) for Different Parameters and Problem Sizes

[Proportions of valid solutions are given in parentheses]

Parameter set	Parameter	Values of q for problem sizes (number of elements) of—				
		$n = 10$	$n = 20$	$n = 30$	$n = 50$	$n = 100$
1	$A = B = 2,\ C = 2,\ D = 1$	0.988 (1.0)	0.960 (1.0)	0.975 (1.0)	0.978 (1.0)	0.987 (1.0)
2	$A = B = 200,\ C = 20,\ D = 50$	1.0 (1.0)	0.999 (1.0)	0.999 (1.0)	0.998 (1.0)	0.998 (1.0)
3	$A = B = 200,\ C = 3,\ D = 50$	1.0 (1.0)	0.999 (1.0)	1.0 (1.0)	1.0 (1.0)	0.999 (1.0)

Another aspect to the comparison between optimization networks and conventional algorithms is the time it takes to solve a problem of a particular size. For example, it takes more than 1 day of processing time on a VAX-11/780 (manufactured by Digital Equipment Corporation) to *simulate* the neural network solving a single 100-city problem. This is actually not

surprising because the simulation involves the numerical solution of 10^4 ODE's for several thousand iterations. However, the Lin-Kernighan algorithm provides an answer (usually better) in about 3 minutes. Furthermore, 100 cities are not even considered an interesting problem size for the TSP. Although an analog hardware implementation of the neural network might solve the same problem in milliseconds, the need for a very large-scale integrated (VLSI) chip with 10^4 operational amplifiers to solve a 100-city TSP is truly questionable. Thus, we do not think that large-scale, classical or NP-complete optimization problems are suitable applications for optimization networks other than as examples or model problems. However, there are certain small-scale, special-purpose, real-time control problems that could benefit from the key characteristics of an ANN hardware implementation: e.g., speed, low weight and power consumption, and built-in fault tolerance.

Thus, our actual objective is not to compete with conventional methods in solving classical optimization problems but to investigate the fault tolerance of the network for special-purpose applications. The above performance assessment is a prerequisite for this investigation. In the next section, we still use the TSP and AP as model problems to demonstrate and to quantify the performance degradation under the presence of simulated faults in the network. Section 6 then describes an application in which an optimization network controls the reconfiguration of a multiprocessor system. There, the fault tolerance of the network is the decisive factor for the operation of the system.

5. Fault Tolerance

Fault tolerance is a qualitative, general term defined as the ability of a system to perform its function according to the specification in spite of the presence of faults in its subsystems. This definition is very unspecific, and a system that is said to be fault tolerant does not necessarily tolerate any number of faults of any kind in any of its subsystems. A specific way to quantify fault tolerance is to determine the performance degradation in the presence of certain faults in certain subsystems, given that some measure of performance exists.

Only relatively few studies in the literature focus explicitly on the fault tolerance of ANN's, and the results are difficult to generalize because of the different models and objectives. For example, Hinton and Shallice (1989) injected faults into a neural network trained to perform a particular linguistic task. They showed that the performance degradation of the network bears a qualitative resemblance to the degraded ability of neurological patients with a certain brain disorder. Petsche and Dickinson (1990) used a special network architecture to investigate a self-repair mechanism that automatically activates spare nodes (neurons) if one of the nodes is inoperable, i.e., permanently inactive ("stuck at 0"). A study that is more closely related to our approach was performed by Belfore and Johnson (1989) who also investigated the effect of faults in an optimization network that solves the TSP. However, they used only a single six-city distribution with single-node faults in their simulations, which is insufficient to draw any statistically meaningful conclusion as we will show below.

According to figure 1, there are only two different components in a hardware implementation of an optimization network: the neuron or active element in the form of an operational amplifier, and passive interconnections in the form of resistors. In the following, we will first consider two types of faults of the active elements that correspond to the highest failure rate. These are commonly called *stuck-at-1* or *stuck-at-0* faults and occur if the output of a unit (amplifier) is permanently pulled to the highest potential or to the lowest (ground) potential, respectively. The fault locations are randomly selected with one important exception: *we do not allow two stuck-at-1 faults to occur within the same row or column.* The reason is that such an event would automatically preclude a valid solution since the permutation matrix allows only one 1 in each row and column. In simulating multiple faults, we study a succession of either stuck-at-1

or stuck-at-0 faults, but not a mixture of both types. We use the same locations for stuck-at-1 and stuck-at-0 faults in order to compare the effect of a different fault type. Otherwise, it would not be possible to tell whether different results are caused by the different locations or by the different fault types. This means that the above exception is also valid for stuck-at-0 faults, although two or more stuck-at-0 faults in the same row or column do not necessarily interfere with a valid solution.

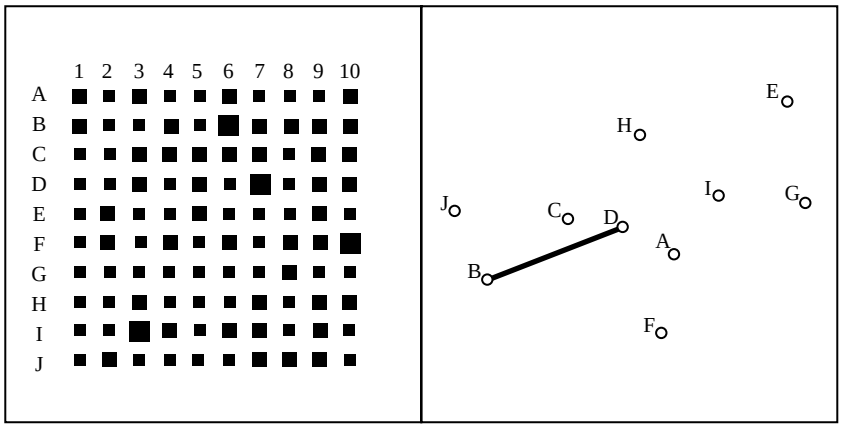
Before we present the results of our large-scale simulations, we want to illustrate the impact of stuck-at-1 faults for several examples. Figures 12 and 13 use the same 10-city TSP examples from section 3.3 to show the effect of 4 stuck-at-1 faults simultaneously present in the network. It can be seen that the networks still converge to a solution; however, the resulting tour is clearly worse than in the fault-free cases of figures 8 and 9. In order to understand these results, it is necessary to recall the definition of a fault in this context. Since we interpret the neuron output as a decision about the position of a city on a tour, a stuck-at-1 fault represents such a decision and thereby predetermines a part of the overall tour. Because of the degeneracy of the TSP problem representation, a single stuck-at-1 fault does not constrain the network at all since the absolute position of a city does not matter. The effect of two simultaneous faults is immediately obvious if the two faults occur in adjacent columns. As shown in figures 12(a) and 13(a), such an event predetermines a link between two cities because the cities are in successive positions on the tour. Figures 12(b) and 13(b) show how this imposed link affects the overall tour.

Surprisingly, this predetermination of parts of a tour by the injected faults does not necessarily lead to a performance degradation. Since the network usually finds a suboptimal solution in the fault-free case, it is conceivable that a lucky combination of fault locations leads to a tour that is actually better than one without any faults. Although these events are rare, we could observe occasional improvements under the presence of multiple faults. Stuck-at-0 faults play a less prominent role because they only *preclude* a city from being in a certain position instead of predetermining it. Thus, the network has even more ways to "work around" those faults, and we would expect a minimal impact from even multiple stuck-at-0 faults.

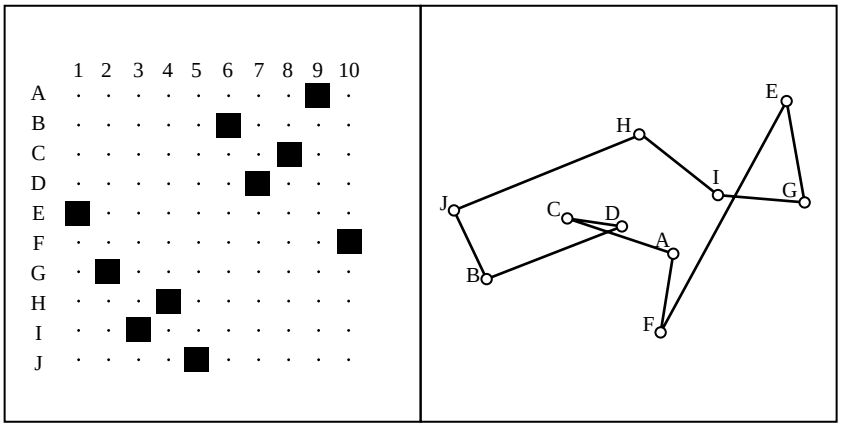
Figure 14 shows the effect of injected stuck-at-1 faults on a network solving the assignment problem. The parameters used for this example are those listed in table 2 (in parameter set 2). The solution shown in figure 14(a) represents the global optimum. Thus, if the best answer is derived under fault-free conditions, any fault can only decrease the performance. Because the AP representation does not have the degeneracy like the TSP, even a single stuck-at-1 fault precludes a convergence to the global solution. Figures 14(b)–(f) illustrate how the multiple-fault locations marked by the shaded squares become part of the solutions and how the network converges to accommodate these constraints.

We analyzed the network solutions in figures 14(b)–(f) by using our conventional algorithm and by taking the faults into account as additional constraints to the problem. Interestingly, the network arrived at the same results, which means that it still found the new global optimum under these fault conditions. Thus, we could define a *conditional performance measure* by viewing the faults as constraints to the problem and assessing the network performance accordingly. Although we can see the obviously unavoidable performance degradation in absolute terms, the conditional performance of the AP network is still optimal. As with the TSP, stuck-at-0 faults preclude a particular solution and have no effect at all unless the fault location coincides with an active unit that is part of the solution; in this case, the network treats the fault as an additional constraint and converges to the best possible solution.

Although the above examples provide some (qualitative) insight into the fault-tolerance characteristics, it is still necessary to substantiate this impression by large-scale simulations in order to obtain more rigorous results. We used the test set of problem instances as defined in the last section and the same parameters that correspond to the results in the second row



(a) Initial state.

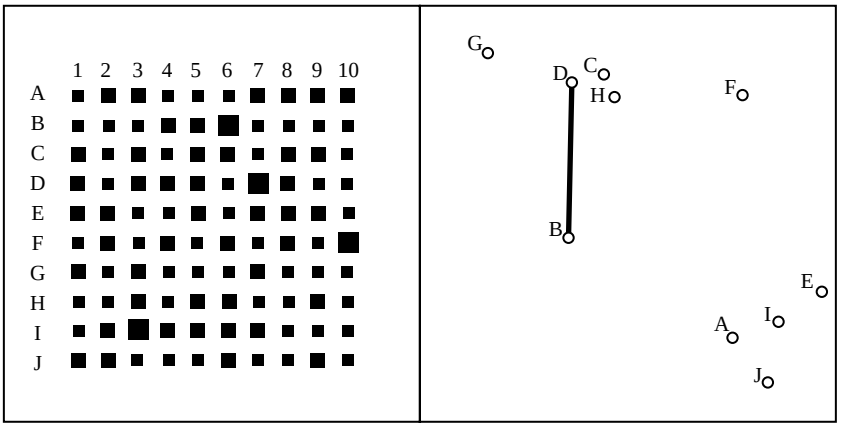


(b) Final result.

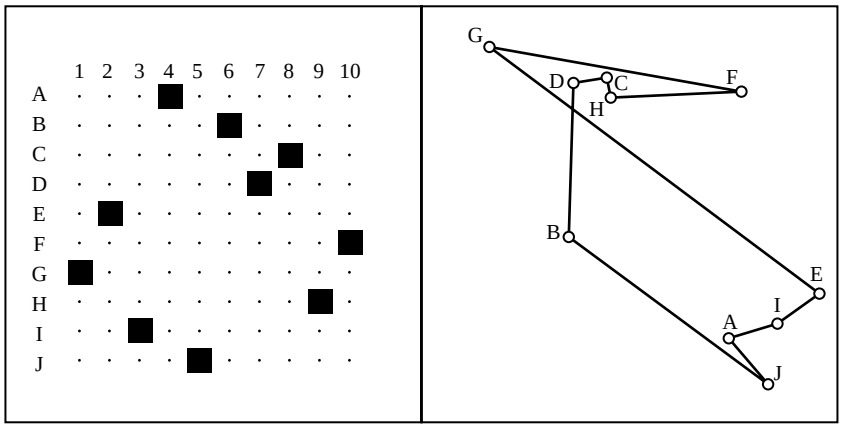
Figure 12. Solution of 10-city problem in figures 7(a) and 8 by network with 4 stuck-at-1 faults (tour length of 3.27). Fault locations are visible during initialization in part (a). Note that the two faults in adjacent columns predetermine a link between cities B and D.

of tables 1 and 2. Only these parameter values were used for the TSP because we regard the consistent convergence to a valid solution in the fault-free case as a prerequisite for any fault-injection experiments. Figure 15 shows the results for different problem sizes. The results confirm our conjecture that stuck-at-0 faults have no effect for the AP and practically no effect for the TSP. In case of the TSP, the injected faults override the random initialization and the network converges without or independent of any initial bias to the same solution. Stuck-at-1 faults result in an almost linear performance degradation for the AP, whereas the redundancy of the TSP representation is reflected in a relatively slower performance decrease as the number of faults increases. When the number of stuck-at-1 faults approaches the number of cities or elements, the performance for both the TSP and the AP approaches zero as in figure 15(a), which corresponds to the random average. This is because the randomly selected fault locations eventually predetermine a random tour. Most importantly, none of our simulations failed to converge to a valid tour because of one or more injected faults.

In another experiment, we studied the effect of connection faults on the performance of an optimization network. Although the failure rate of a simple resistive connection is orders of



(a) Initial state.

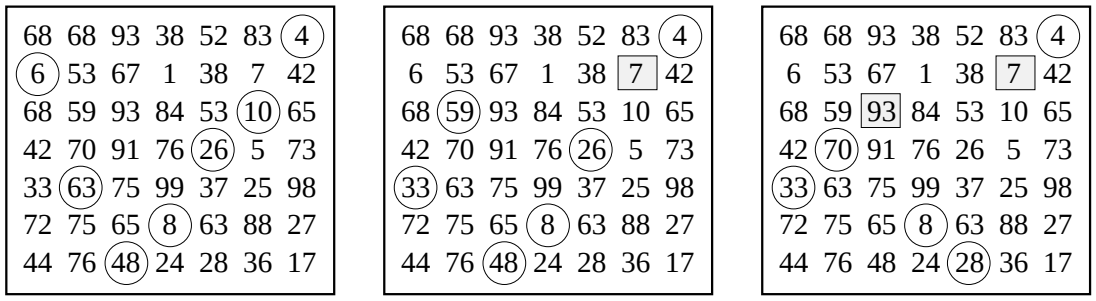


(b) Final result.

Figure 13. Solution of 10-city problem in figures 7(b) and 9 by network with 4 stuck-at-1 faults (tour length of 3.77). Fault locations are visible during initialization in part (a). Note that the two faults in adjacent columns predetermine a link between cities B and D.

magnitude less than that of an operational amplifier, the large number of connections (e.g., $2n^3 - 2n^2$ connections for an n-element AP compared with n^2 neurons) increases the overall probability of such a fault. The failure of a connection with the resistance R leads either to a short circuit ($R = 0$) or to an open connection ($R = \infty$). Because the failure rate of a connection short circuit is far less than the rate of an open connection, we simulated only the latter fault type. In order to limit the number of required simulations, we used only a network solving the AP for this experiment because this network exhibited the best performance and greatest fault tolerance in our previous studies.

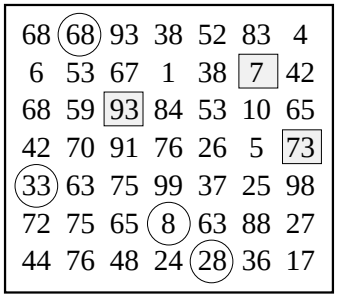
Figure 16 shows the resulting performance degradation of an ANN solving a 10-element AP for up to 50 simultaneous open connections. The parameters for the AP network are the same as in the previous fault-injection runs. The locations for the connection faults were randomly selected. For each fault scenario we ran 50 different problem instances, and figures 16(a) and (b) show the average, worst, and best performance for the two different values of the parameter $D = 50$ and $D = 120$. The parameter D is a factor multiplied by the cost values according to



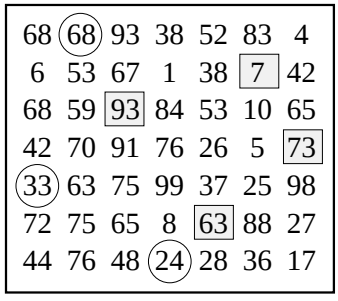
(a) No faults; $c = 165$.

(b) One stuck-at-1; $c = 185$.

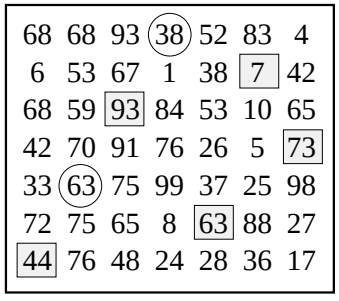
(c) Two stuck-at-1; $c = 243$.



(d) Three stuck-at-1; $c = 310$.



(e) Four stuck-at-1; $c = 361$.



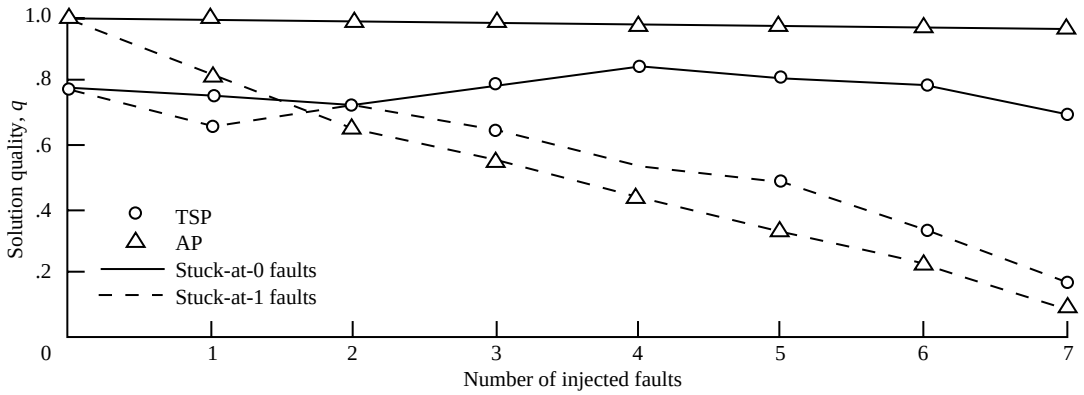
(f) Five stuck-at-1; $c = 381$.

Figure 14. Effect of up to five multiple stuck-at-1 faults on network solving assignment problem of size $n = 7$. Cost matrix is shown with circled elements indicating network solutions (neurons that converged to 1) and shaded squares indicating fault locations. Note that parts (b)–(f) are still optimal solutions under additional constraints imposed by faults.

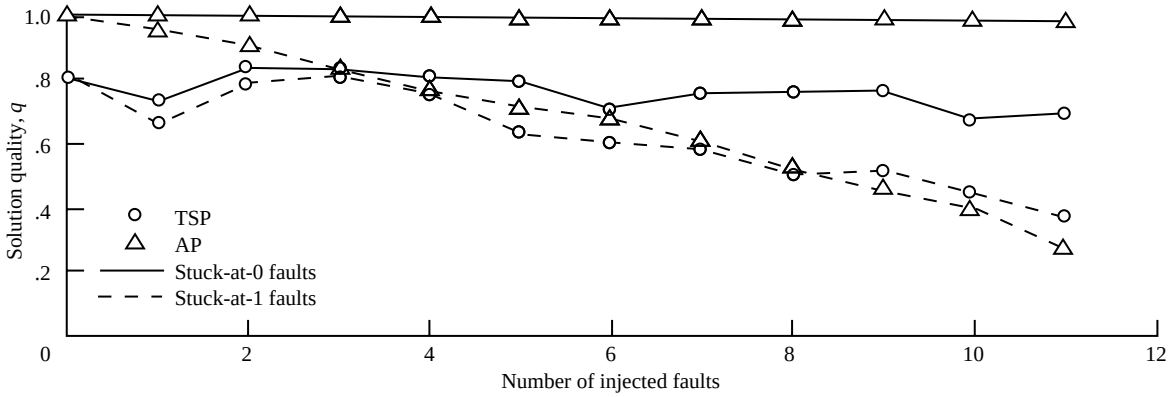
equation (15), and a large value of D enforces solutions with better quality. This is reflected by figure 16(b) which shows a better average quality as well as a lower variation in the quality of the best and worst solutions compared with figure 16(a). This high variation in figure 16(a) is again a reminder of how much the results depend on the chosen problem instance and that the study of a single instance as in Belfore and Johnson (1989) can lead to grave misinterpretations.

Although the performance results suggest that a higher value of D would be desirable, there is a trade-off shown in figure 16(c). Surprisingly, although none of the "stuck-at" faults led to an invalid solution, we do observe invalid solutions for some problem instances after a certain number of open connections. Figure 16(c) shows the percentage of valid solutions, and it can be seen that a lower value of D tolerates more faults before the first case of an invalid solution occurs. We have already seen this trade-off between consistently valid and high-quality solutions in the fault-free cases of section 4, and it is very interesting to observe that the same effect plays an important role with respect to the fault tolerance. Because an invalid solution is the worst case and equivalent to a total system failure, a smaller value[4] of D is obviously preferable for the AP, especially since it does not affect the fault-free performance at least for the cases shown in figures 16(a) and (b). However, for a value of $D > 120$, we could also observe some invalid results

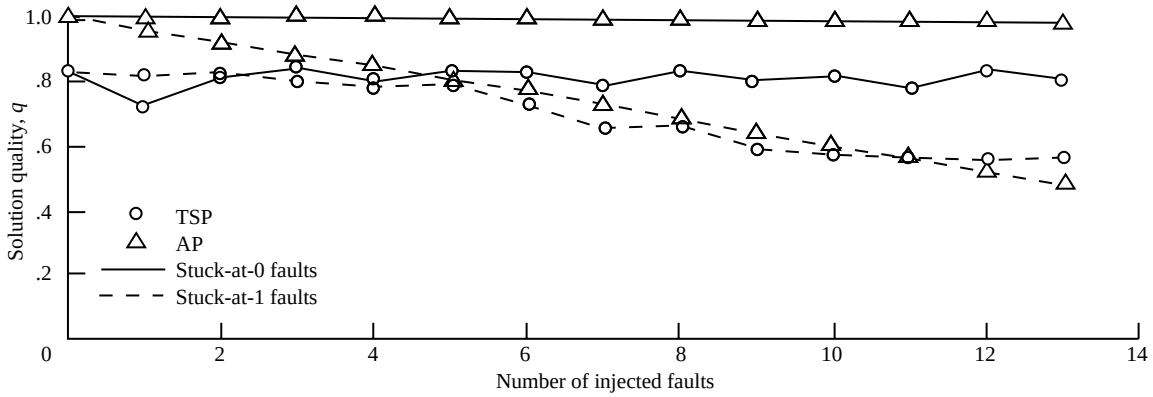
[4] Unfortunately, these qualitative recommendations about the relative size of parameter values do not necessarily hold in general. Since no theory exists to prescribe parameter values for optimization networks, optimal values have to be determined experimentally for each problem.



(a) $n = 10$.

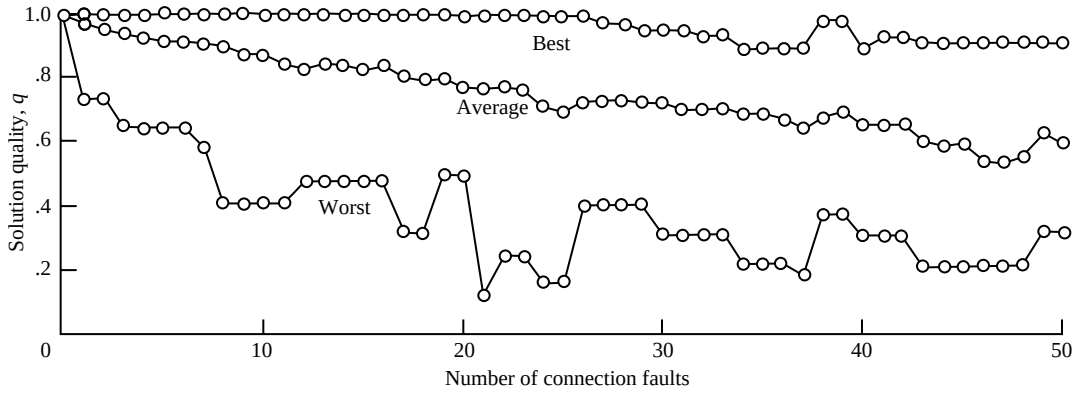


(b) $n = 20$.

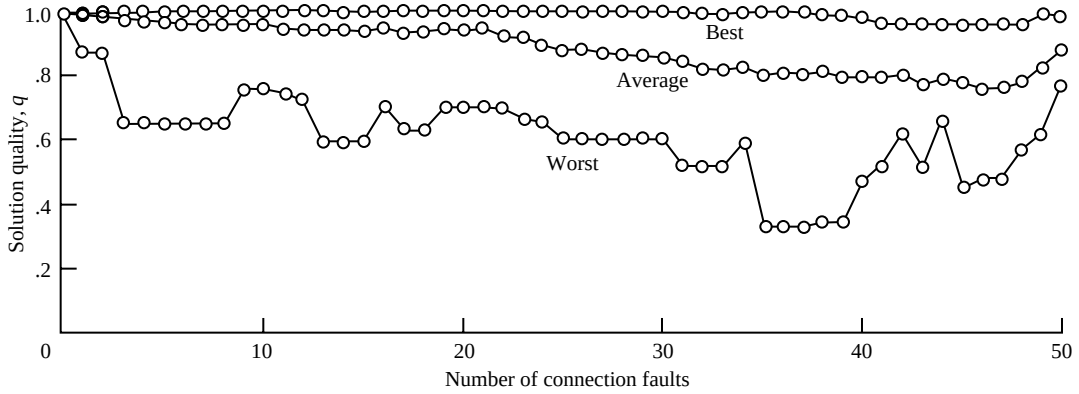


(c) $n = 30$.

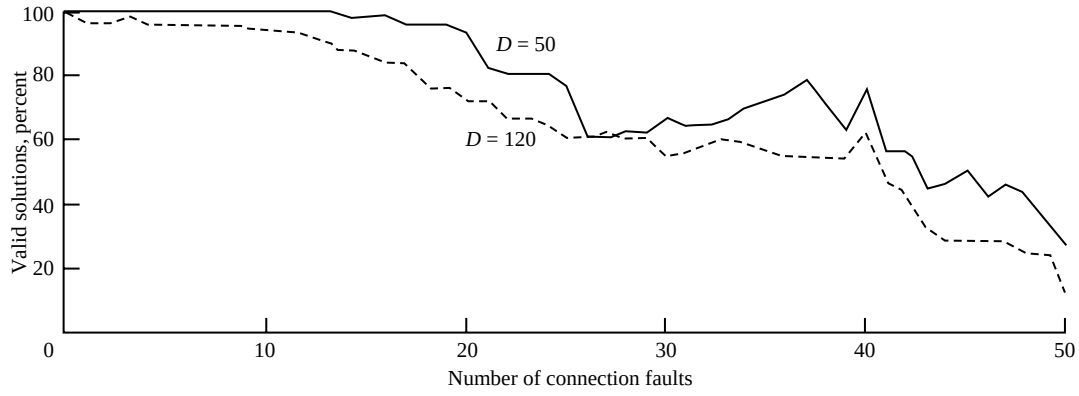
Figure 15. Performance degradation of an ANN solving traveling salesman problem (TSP) and assignment problem (AP) after injections of stuck faults for different problem sizes. Values are averages over 10 different problem instances for each size with additionally 10 different random initializations each for the TSP.



(a) $n = 10; D = 50$.



(b) $n = 10; D = 120$.



(c) Percentage of valid solutions for parts (a) and (b).

Figure 16. Performance degradation of an ANN solving assignment problem (AP) after multiple connection failures (open connections). Values in parts (a) and (b) are the best, worst, and average performance of 50 different problem instances, and values in part (c) indicate how many of the 50 runs for each fault scenario converged to a valid solution.

in the fault-free case. This shows that the "quality-validity trade-off" is a general phenomenon and that connection faults only increase the likelihood of invalid solutions.

In summary, we have demonstrated that optimization networks exhibit a surprising degree of fault tolerance which is achieved without the explicit use of redundant components. Because the fault-tolerance characteristics are inseparable from the functional characteristics, we can say that the fault tolerance of the ANN is built-in or *inherent*. However, when we make a statement about the fault tolerance, we implicitly assume a failure condition or *failure criterion* of the system, which is the threshold below which it can no longer perform its function according to the specification. For example, consider the AP network that always generates the global optimum under fault-free conditions. If we specify this as the only acceptable performance level, then any stuck-at-1 fault that causes the network to generate a good but suboptimal answer is not acceptable and, with respect to this fault type, the network is not fault tolerant at all. On the other hand, if we specify a solution quality of 0.8 as the acceptable performance threshold, then an AP network of size $n = 30$ can tolerate (on the average) five stuck-at-1 faults and an even larger number of stuck-at-0 or connection faults. Thus, the degree of fault tolerance depends on our definition of acceptable performance.

The above discussion suggests an application domain for optimization networks where it is not necessarily important to generate the best possible solution to an optimization problem, but where a reasonably good answer has to be obtained fast and reliably. In the next section we present an example of such an application with the network performing a critical real-time task as a component of a fault-tolerant multiprocessor system.

6. Application of an ANN for the Task Allocation in a Distributed Processing System

In the following discussion we will investigate the application of an optimization network in the context of a distributed processing system that operates under hard real-time constraints and has to meet very high reliability requirements. An example of such a system is the Software-Implemented Fault-Tolerance (SIFT) computer used by NASA as an experimental vehicle for fault-tolerant systems research (Palumbo and Butler 1986). The SIFT architecture can accommodate up to eight processors in a fully distributed configuration with a point-to-point communication link between every pair of processors. It can be used, for example, to execute real-time flight control tasks as part of an aircraft control system. Because the system operates in a distributed fashion, each processor executes a certain number of tasks according to a predetermined task-to-processor allocation table.

The architecture achieves an extreme fault tolerance by its capability to detect and isolate possible hardware faults. The isolation of a defective processor requires a reconfiguration of the system and a reallocation of all tasks among the remaining processors. Thus, it is not the initial task allocation but the reallocation of tasks after a processor failure that is time critical and has to be performed by a highly reliable mechanism. The use of lookup tables for the reallocation has the disadvantage that the number of combinations of tasks and processors is very large for even moderately sized systems (Bannister and Trivedi 1988) and grows exponentially after multiple processor failures. Although it is possible to use conventional algorithms to solve the problem, these methods are often computationally too expensive because of the hard real-time constraints and require an undesirable overhead because the algorithms have to be executed in a distributed environment without any hierarchical control.

Since finding that the best allocation of tasks among the processors can be formulated as a constrained optimization problem, we will demonstrate how an optimization network can be used to solve this problem. The distributed system considered here resembles a simplified version

of the SIFT computer and is based on a model described by Bannister and Trivedi (1988) in which a conventional heuristic algorithm is used to solve this task allocation problem. We will later use this algorithm as a benchmark to assess the ANN performance. The system has to execute n tasks and consists of m identical processors. Each task is replicated into r clones that are executed by different processors and submitted to a majority voter in order to detect and mask possible hardware failures. By assuming periodic real-time tasks for a typical flight control system, the number of instructions per execution of task j, the frequency of execution, and the execution rate of the processor determine the load that a certain task places on a processor, which is called the utilization z_j of task j. A particular allocation can be described by a variable V_{ij} with $V_{ij} = 1$ if task j is scheduled on processor i, and $V_{ij} = 0$ otherwise. Then, the variable $p_i = \sum_j z_j V_{ij}$ represents the overall load or utilization of processor i under the allocation V_{ij}.

The task allocation has to observe the constraint that each task must be executed by exactly r different processors in order to allow a majority vote. Additionally, the allocation should be done in a way that achieves at least an approximate load balancing among the processors. A load balancing in a distributed processing system is obviously desirable, and Bannister and Trivedi (1988) discuss several reasons why an imbalance potentially decreases the reliability of the system. They also show that minimizing the sum of the squared processor utilizations $\sum_i p_i^2$ also minimizes the statistical variance of the p_i variables, which is a direct measure of the imbalance. We further assume that there are enough processors to accommodate a (balanced) assignment without capacity or scheduling violations.

The task allocation problem (TAP) is represented by an optimization network consisting of a two-dimensional array of $m \times n$ neurons or elements in which the output V_{ij} of an element is bounded between 0 and 1 and corresponds to the hypothesis that task j is assigned to processor i. Figure 17 illustrates this problem representation for an example in which 10 triplicated tasks are allocated to 5 processors. In order to map the task allocation problem onto the network, it has to be expressed as a function whose minima correspond to (local) solutions of the problem. With the above definitions, we can define the following energy function

$$E_{\mathrm{TAP}} = \frac{A}{2} \sum_{j=1}^{n} \left(\sum_{i=1}^{m} V_{ij} - r \right)^2 + \frac{B}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} V_{ij} \left(1 - V_{ij} \right) + \frac{D}{2} \sum_{i=1}^{m} \left(\sum_{j=1}^{n} z_j V_{ij} \right)^2 \tag{26}$$

The first term in equation (26) has a minimum if the constraint is met (i.e., each task is executed by exactly r processors), the second term forces the outputs to converge to either 0 or 1, and the third term represents the cost function to be minimized. Mapping equation (26) onto the energy function (eq. (12)) yields the following values for the interconnections and the external current

$$\left.\begin{aligned} T_{ij,lk} &= -A\delta_{jk} + B\delta_{il}\delta_{jk} - Dz_j z_k \delta_{il} \\ I_{ij} &= Ar - \frac{B}{2} \end{aligned}\right\} \tag{27}$$

and the equations of motion

$$\begin{aligned} C_{ij} \frac{du_{ij}}{dt} = &- \frac{u_{ij}}{R_{ij}} - A \sum_l V_{lj} + BV_{ij} \\ &- Dz_j \sum_k z_k V_{ik} + Ar - \frac{B}{2} \end{aligned} \tag{28}$$

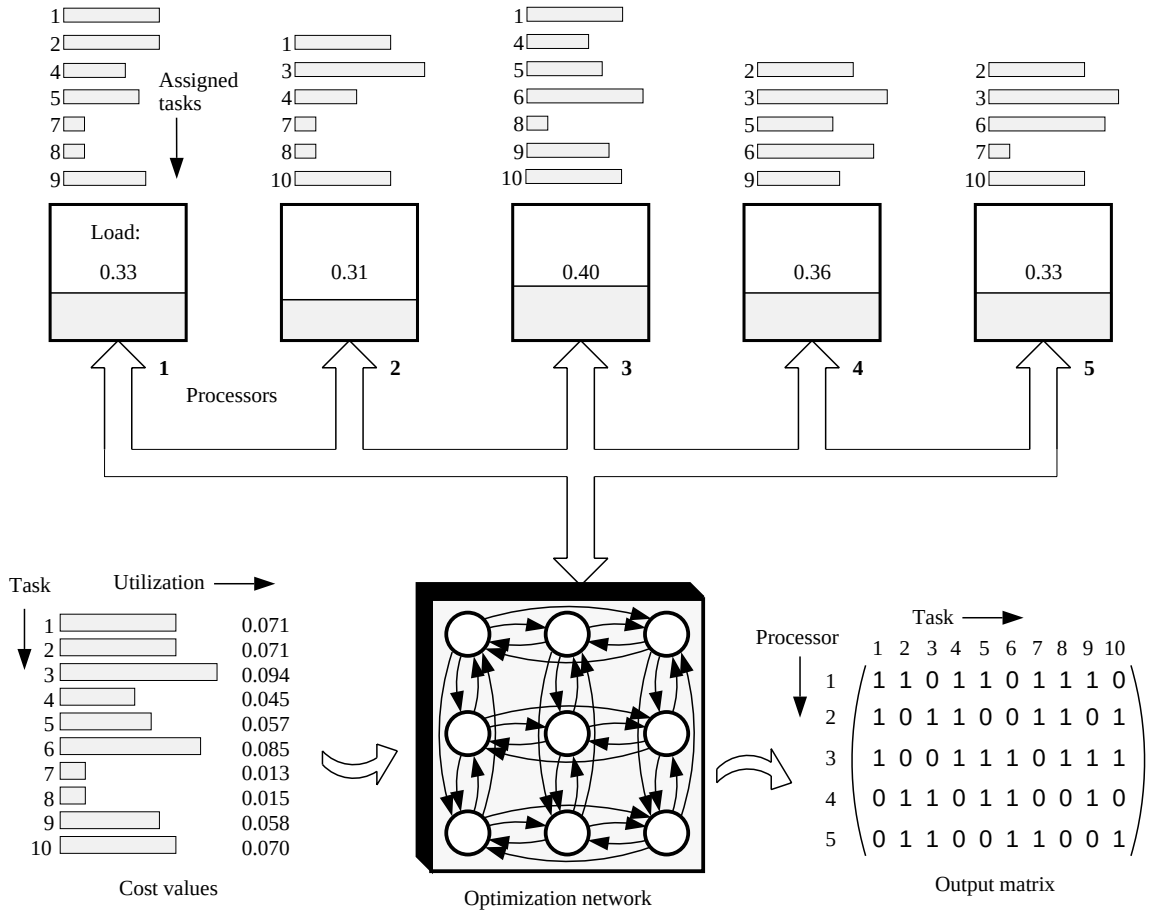


Figure 17. Example of allocation of tasks to processors generated by optimization network. Note that each task has to be executed by exactly three different processors while an approximate load balancing of processors should be achieved.

We used the parameter values $A = 75$, $B = 5$, and $D = 350$ as well as $\lambda = 25$ and $u_s = 0$ for the transfer function (eq. (1)). Although there are only three parameters, we used D as the third parameter because we have previously associated D with the cost function of the problem. Our simulations are performed for different data sets with task utilizations z_j randomly generated from a uniform distribution between 0.01 and 0.10. Because of the quadratic cost function in equation (26), the cost values z_j are part of the interconnections and the external current is constant. Thus, this problem is similar to the TSP and requires a random initialization to overcome the unstable equilibrium point at $u_{ij} = 0$. We used the initial values $V_{ij} = 0.5 + \delta$ with small, uniform noise ($-10^{-7} \leq \delta \leq 10^{-7}$). The equations of motion (eq. (28)) were solved by Euler's method with a step size $\Delta t = 2 \times 10^{-5}$ and required an average of 5000 iterations to converge.

At this point, we can simulate the network and successfully solve the TAP as shown in figure 17 with a performance that is comparable to the TSP network, but this is not the actual task in this application. What is required is a *reallocation* of tasks *after a processor failure*. Therefore, the network has to be provided with the information about which processor has failed. Furthermore, it has to implement this information as an additional constraint before solving the problem. For example, the unavailability of a processor k can be represented by enforcing $V_{kj} = 0$ for all tasks j; that is, no tasks can be assigned to processor k. This additional constraint could be implemented either by external currents of sufficient strength to inhibit all neurons in row k

or by switches connecting the outputs of all neurons in row k to 0 (ground potential). Although the latter method seems to be somewhat crude, it actually has the advantage that a possible stuck-at-1 hardware fault of a neuron in that row is overridden by the external switch. Producing this short circuit at the outputs is equivalent to our stuck-at-0 fault injections in the last section; there we showed that the network indeed treats this condition as an additional constraint to the optimization problem. Figure 18 illustrates the process of reallocation after a processor failure by using the same example shown in figure 17.

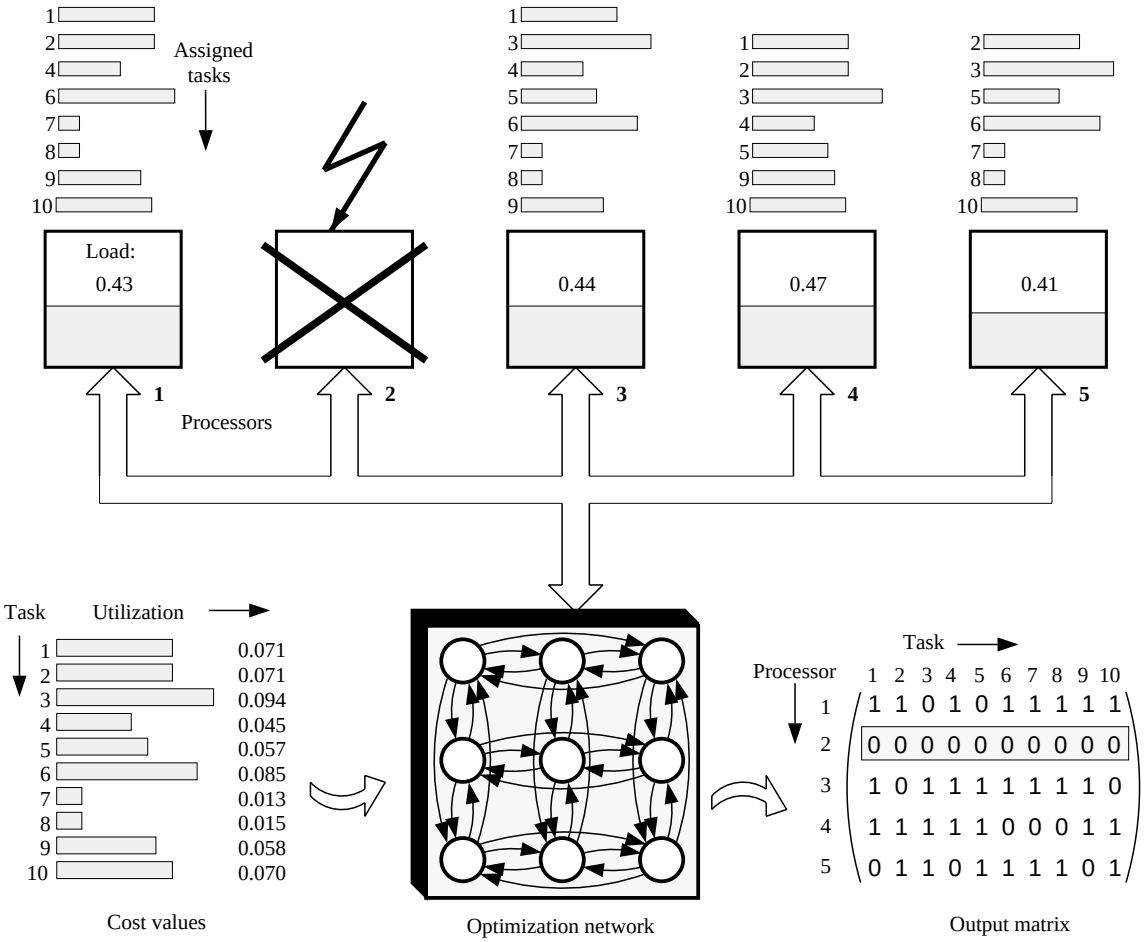
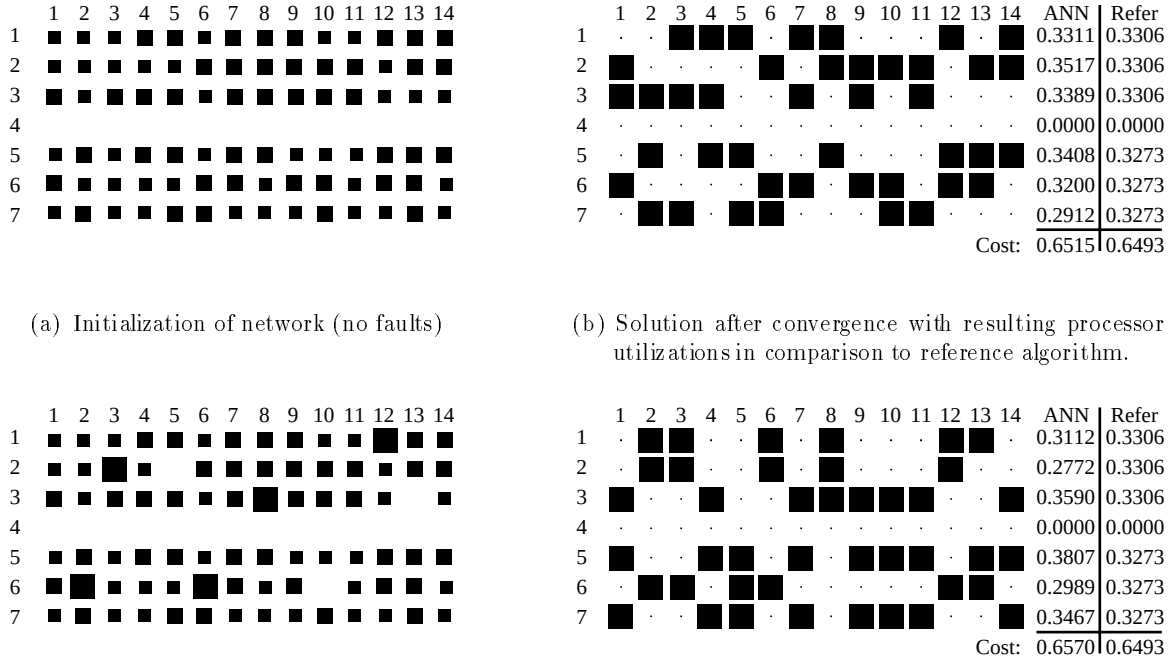


Figure 18. Example of reallocation of tasks after a processor failure. Optimization network generates new allocations by observing the constraints and by approximately balancing the load of the processors.

The network is obviously a critical component of the system because a network failure would prevent the reconfiguration of the system after a processor failure, which leads to a total system failure. Thus, the fault tolerance of the ANN becomes a crucial characteristic. We tested the fault tolerance again by simulating stuck-at-0 and stuck-at-1 faults in randomly selected locations. Figure 19 illustrates the operation and convergence of the network for the example of a system with $m = 7$ processors and $n = 14$ tasks where each task has to be executed by three different processors ($r = 3$). Figure 19(a) shows the initialization of the (fault-free) network for a scenario in which processor 4 has failed, which is reflected by an output value of zero for all neurons in row 4. Figure 19(b) indicates the result after convergence with tasks 2, 3, and 6 assigned to processor 1, tasks 3, 5, and 7 assigned to processor 2, etc. The load-balancing performance of the ANN is also illustrated in figure 19(b) which lists the processor utilizations resulting from

the ANN solution in comparison with a simple, heuristic reference algorithm (Bannister and Trivedi 1988). As can be seen from the cost values listed at the bottom, which are the sum of the squares of the processor utilizations, the ANN is outperformed by the algorithm although the difference of the values is only of the order of 1 percent. However, as we stated earlier, an approximate load balancing is sufficient in this case as long as the solution can be obtained fast and reliably.



(a) Initialization of network (no faults)

(b) Solution after convergence with resulting processor utilizations in comparison to reference algorithm.

(c) Initialization (five stuck-at-1 and three stuck-at-0 faults injected into network).

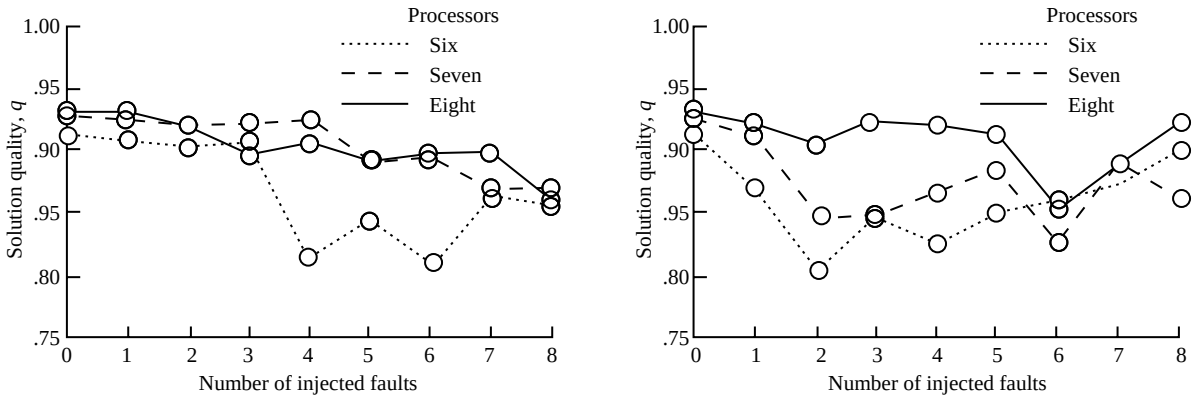
(d) Solution after convergence under presence of injected faults.

Figure 19. Illustration of operation and convergence of network generating a task allocation after failure of processor 4 ($m = 7,\ n = 14,$ and $r = 3$).

The latter requirement is illustrated in figure 19(c), which shows the initialization of the network for the same scenario, but now with *eight* faults simultaneously present in the network. The fault locations of five stuck-at-1 and three stuck-at-0 faults are clearly recognizable after the initialization. Figure 19(d) shows the results after convergence, and we can observe the same phenomenon that the faults do not impair the convergence but act as additional constraints of the problem. According to the cost value in figure 19(d), the performance is only slightly worse than in the fault-free case.

Since the performance of the ANN varies considerably for different random initializations and different input data, it is necessary to evaluate the average performance over a sufficient number of problem instances in order to obtain a statistically relevant assessment. We simulated a system with $m = 8$ processors and $n = 24$ triplicated tasks ($r = 3$), which requires a network of 192 neurons. Seven different test sets of random task utilizations were generated. The network was simulated with seven different initializations for each test set. The solution quality q was used to assess the performance where values for c_{opt} were obtained from the heuristic algorithm in Bannister and Trivedi (1988). Figure 20 demonstrates the performance degradation for up to

eight injected stuck-at-0 or stuck-at-1 faults. The number of processors refers to the remaining number of available processors in the system. For example, if the distributed system consists initially of eight processors, then $m = 7$ refers to the operation of the network after a failure of one processor with the neurons in the corresponding row switched to zero. Note that the solution quality in figure 20 is plotted in the small range from 0.75 to 1.00, which magnifies the variations. As expected, the performance is very similar to the TSP because both use a quadratic cost function.



(a) Stuck-at-0 faults.

(b) Stuck-at-1 faults.

Figure 20. Performance degradation of ANN allocating $n = 24$ triplicated tasks ($r = 3$) to $m = 8$, 7, and 6 processors.

The results in figure 20 confirm the qualitative observation in figure 19 that the ANN exhibits an extreme fault tolerance compared with conventional systems. Since the faults are randomly located and act as additional constraints of the problem, it is possible that one or more faults accidentally dictate a better solution than the network would have found without faults. This explains the occasional *performance increase* after fault injection and the nonmonotonic characteristic of the performance degradation. Of course, this is only possible because of the suboptimal performance of the ANN in the fault-free case. It is also important to note that none of the simulations converged to an invalid solution or to a solution that violates the capacity constraint $p_i < 1$, although the latter was not explicitly enforced. An event that would lead to an invalid solution can occur only if there are more than r stuck-at-1 faults in the same column, thus assigning a task to more than r processors and violating the constraints. If the faults occur at random locations and if the failure rate of a stuck-at-1 fault is known for a particular hardware implementation, then this scenario can be used to estimate an upper bound for the reliability of the ANN.

7. Concluding Remarks

The objective of our investigation was to explore the fault-tolerance characteristics of a particular neural network type and to show how these networks might be used in certain critical applications. First, we described the principle of operation of these networks and showed how they can be used to solve optimization problems. The operation and the performance of the network was first illustrated for two examples of classical optimization problems, the assignment problem and the traveling salesman problem. With an analog hardware implementation of the neural network in mind, the fault tolerance was simulated by subjecting the "neurons" implemented as operational amplifiers to multiple "stuck-at-1" and "stuck-at-0" faults.

We have demonstrated that the fault tolerance is an inherent characteristic of this type of neural network and that the injected faults are treated by the network as additional constraints to the problem. Although conventional systems often break down completely after a single fault, the network exhibits a graceful performance degradation even after multiple injected faults. This characteristic can be exploited and a fault-tolerant neural network integrated on a single analog very large scale integrated (VLSI) chip might perform a critical task that would otherwise require a redundant microprocessor system with specially tested software.

As an example of a promising application, we used the neural network as a critical component of a fault-tolerant, distributed processing system. The failure of a processor requires a reconfiguration of the system and a reallocation of all tasks among the remaining processors. This task allocation has to observe certain constraints and should at least approximately balance the load of the processors. We showed how a neural network can solve this problem and demonstrated the robustness of the network by injecting simulated faults. Our results indicate that the network can indeed perform this task reliably and that even multiple faults do not impair the ability of the network to generate an answer with only slightly degraded performance. The limit of the fault tolerance of the network is problem dependent and is determined by certain scenarios of multiple faults that would lead to a violation of the constraints, such as, for example, more than three stuck-at-1 faults in the same column. Such fault combinations are explicitly excluded in our fault-injection experiments since they would obviously preclude a valid solution. With known failure rates and faults occurring at random locations, these worst-case scenarios can be used to estimate an upper bound for the reliability of the neural network.

In summary, we think that applications exist for the type of neural network described in this paper that can take advantage of the speed, low weight, low power consumption, and fault tolerance of future hardware implementations. However, in most cases, the actual performance of the network does not reach the performance of the best available, conventional optimization algorithm. Thus, the neural network approach is best suited to certain real-time applications that do not necessarily require the absolute best answer, but where it is necessary to generate an approximate answer fast and reliably. The characteristic of a graceful performance degradation without additional redundancy is especially interesting for long-term unmanned missions where component failures have to be expected but no repair or maintenance can be provided.

NASA Langley Research Center
Hampton, VA 23665-5225
February 11, 1992

8. Appendix A

Equations of Motion

In this section we will derive the equations that govern the dynamical behavior of the network shown in figure A1 (which was presented earlier as fig. 1). The symmetry of the network simplifies the analysis, and it is sufficient to determine the equations for a particular (but arbitrary) unit i.

Figure A2 shows an equivalent circuit diagram for such a unit i in which ideal voltage sources represent the feedback from all other units including unit i itself. The nonlinear relationship $V_i = g(u_i)$ between input and output of a unit can be expressed by the sigmoidal function as described in section 2, but it is not required for the following analysis.

The simple circuit in figure A2 can be analyzed by applying Kirchhoff's current law

$$\sum_{j=1}^{n} i_j + I_i = i_r + i_C \tag{A1}$$

and Kirchhoff's voltage law

$$V_j = \frac{i_j}{T_{ij}} + u_i \tag{A2}$$

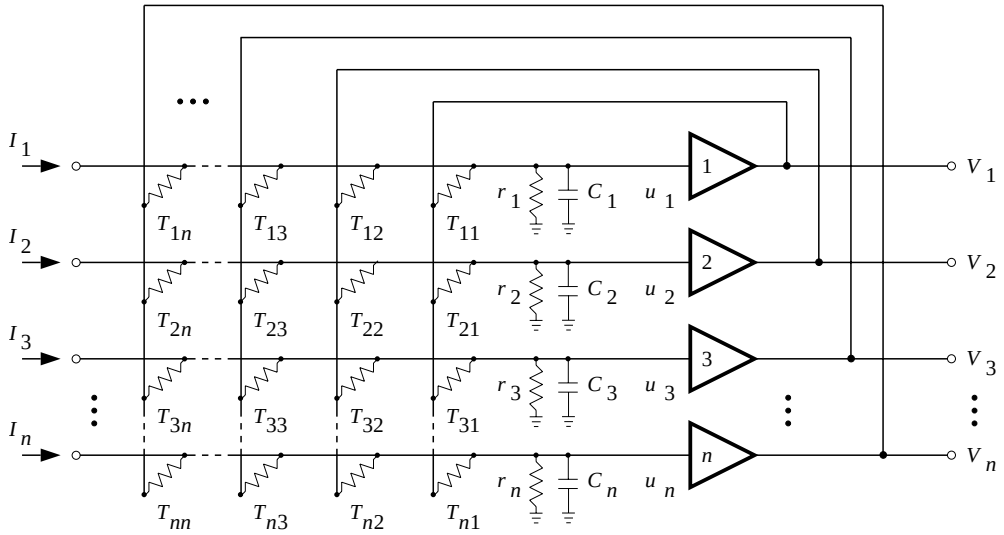


Figure A1. Circuit diagram of optimization network according to Hopfield (1984). Note that negative feedback can be realized by connecting positive conductances T_{ij} to negative output $-V_i$ of unit (not shown in this figure). (This figure, which was presented earlier as fig. 1, is repeated here for the reader's convenience.)

Solving equation (A2) for i_j and combining equations (A1) and (A2) results in

$$\sum_{j=1}^{n} \left(T_{ij} V_j - T_{ij} u_i\right) + I_i = i_r + i_C \tag{A3}$$

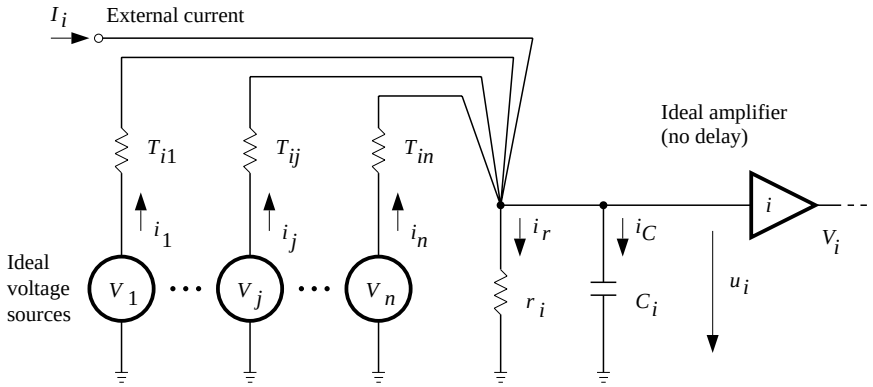


Figure A2. Equivalent circuit diagram of network in figure A1 for one particular unit i.

By substituting the relations $i_r = u_i/r_i$ and $i_C = C(du_i/dt)$ into equation (A3), it follows that

$$C_i \frac{du_i}{dt} + \frac{u_i}{r_i} = -u_i \sum_{j=1}^{n} T_{ij} + \sum_{j=1}^{n} T_{ij} V_j + I_i \tag{A4}$$

After some final rearrangement, we get the "equations of motion"

$$C_i \frac{du_i}{dt} = -u_i \left(\frac{1}{r_i} + \sum_{j=1}^{n} T_{ij} \right) + \sum_{j=1}^{n} T_{ij} V_j + I_i \tag{A5}$$

The parallel combination of the input resistance r_i and all the conductances T_{ij} connected to unit i can be expressed as a single resistance R_i with

$$\frac{1}{R_i} = \frac{1}{r_i} + \sum_{j=1}^{n} T_{ij} \tag{A6}$$

The product of R_i and C_i is often referred to as the time constant τ_i of the equivalent circuit diagram in figure A2. An identical time constant for each unit i would require $C_i = C$ and $R_i = R$ for all units i. The latter condition might be difficult to achieve in practice if the parallel combination of the weights in equation (A6) results in different values for each unit i. In this case, each individual value for r_i would have to be chosen in a way that compensates for these variations.

Also of importance is that the time constant τ_i describes the convergence of the input voltage u_i of unit i. Because of the potentially very high gain of the transfer function $V_i = g(u_i)$, the output V_i might saturate very quickly. Thus, even if the input u_i is still far from reaching its equilibrium point, the output V_i might already be saturated; and by observing only V_i, it might *appear* as if the circuit had converged in merely a fraction of its time constant τ_i.

9. Appendix B

The Energy Function

The stability of the neural network in figure A1 can be proven by considering the following Liapunov or "energy" function:

$$E = -\frac{1}{2}\sum_i \sum_j T_{ij} V_i V_j - \sum_i V_i I_i + \sum_i \frac{1}{R_i} \int_{g(0)}^{V_i} g^{-1}(V)\, dV \tag{B1}$$

With $V_i = g(u_i)$ denoting the sigmoidal transfer function between input u_i and output V_i of element i, the third term in equation (B1) represents an integral over the inverse of this transfer function. Two examples of sigmoidal transfer functions, their inverses, and values for the integral are illustrated in figure B1. For example, with the transfer function

$$V_i = \frac{1}{2}[1 + \tanh(2\lambda u_i)]$$

and its inverse

$$u_i = \frac{1}{\lambda}[\operatorname{arctanh}(2V_i - 1)]$$

the integral term in equation (B1) can be written as

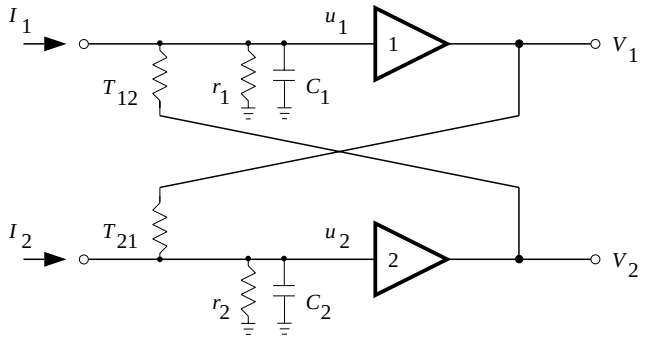
$$\sum \frac{1}{R_i} \int_{g(0)}^{V_i} g^{-1}(V)\ dV = \frac{1}{\lambda}\sum_i \frac{1}{R_i} \int_{0.5}^{V_i} \operatorname{arctanh}(2V - 1)\, dV \tag{B2}$$

The integral term in equation (B2) vanishes in the so-called *high gain limit* with $\lambda \to \infty$. As shown in figure B1(a3), the value of the integral is zero at $V_i = 0.5$ and rises sharply as V_i approaches either 0 or 1. For practical purposes with moderately high gain values λ, the integral term in equation (B2) can be neglected and plays a role only in establishing "energy walls" that represent the borders of the hypercube in V-space ($0 < V_i < 1$ for all units i).

Sketch A presents a network with two neurons mutually interconnected by negative feedback ($T_{ij} < 0$). This is the simplest possible case of an optimization network ($n = 2$) and actually represents the bistable memory element known as *Flip-Flop*. This Flip-Flop is used to demonstrate the shape of the energy function (eq. (B1)), which is shown in figure B2 for different values of λ. If the gain is too small (fig. B2(a)), then the integral term dominates as V_i approaches 0 or 1 and prevents the occurrence of minima at the corners of the space. Instead, the system behaves like a linear system with only one stable equilibrium point at $V_i = 0.5$ ($u_i = 0$). In figure B2(b), two stable states with very shallow minima can be identified because of the higher gain. Although the gain in figure B2(c) is still relatively small with $\lambda = 25$, this case already constitutes the high gain limit. It can be seen that the minima of the energy function are formed where the descending energy surface meets the wall of the cube caused by the integral term in equation (B2). Figure B2(c) also illustrates the unstable equilibrium point at $V_i = 0.5$ ($u_i = 0$). An initialization of the system with identical values for all u_i or V_i, respectively, leads to a movement to the unstable point at the center.

In order to prove the stability of the network, it is necessary to show that the energy function of equation (B1) is indeed a Liapunov function for the system. The time derivative of equation (B1) can be calculated using the chain rule

$$\frac{dE}{dt} = \sum_i \frac{\partial E}{\partial V_i} \frac{dV_i}{dt} \tag{B3}$$



Sketch A

with

$$\frac{\partial E}{\partial V_i} = -\frac{1}{2}\sum_j \left(T_{ij} + T_{ji}\right) V_j - I_i + \frac{u_i}{R_i} \tag{B4}$$

The following steps of the proof require that the condition $T_{ij} = T_{ji}$ be met. Thus, assuming a symmetric connectivity, equation (B3) can be written as

$$\frac{dE}{dt} = \sum_i \frac{dV_i}{dt}\left(-\sum_j T_{ij}V_j - I_i + \frac{u_i}{R_i}\right) \tag{B5}$$

The term in parentheses in equation (B5) is identical to the negative right-hand side of the equations of motion of the network

$$C_i\frac{du_i}{dt} = -\frac{u_i}{R_i} + \sum_j T_{ij}V_j + I_i \tag{B6}$$

By substituting equation (B6) into equation (B5), we can write

$$\frac{dE}{dt} = -\sum_i C_i \frac{dV_i}{dt}\frac{du_i}{dt} \tag{B7}$$

With the relation $dV_i/dt = (dV_i/du_i)(du_i/dt)$, it follows from equation (B7) that

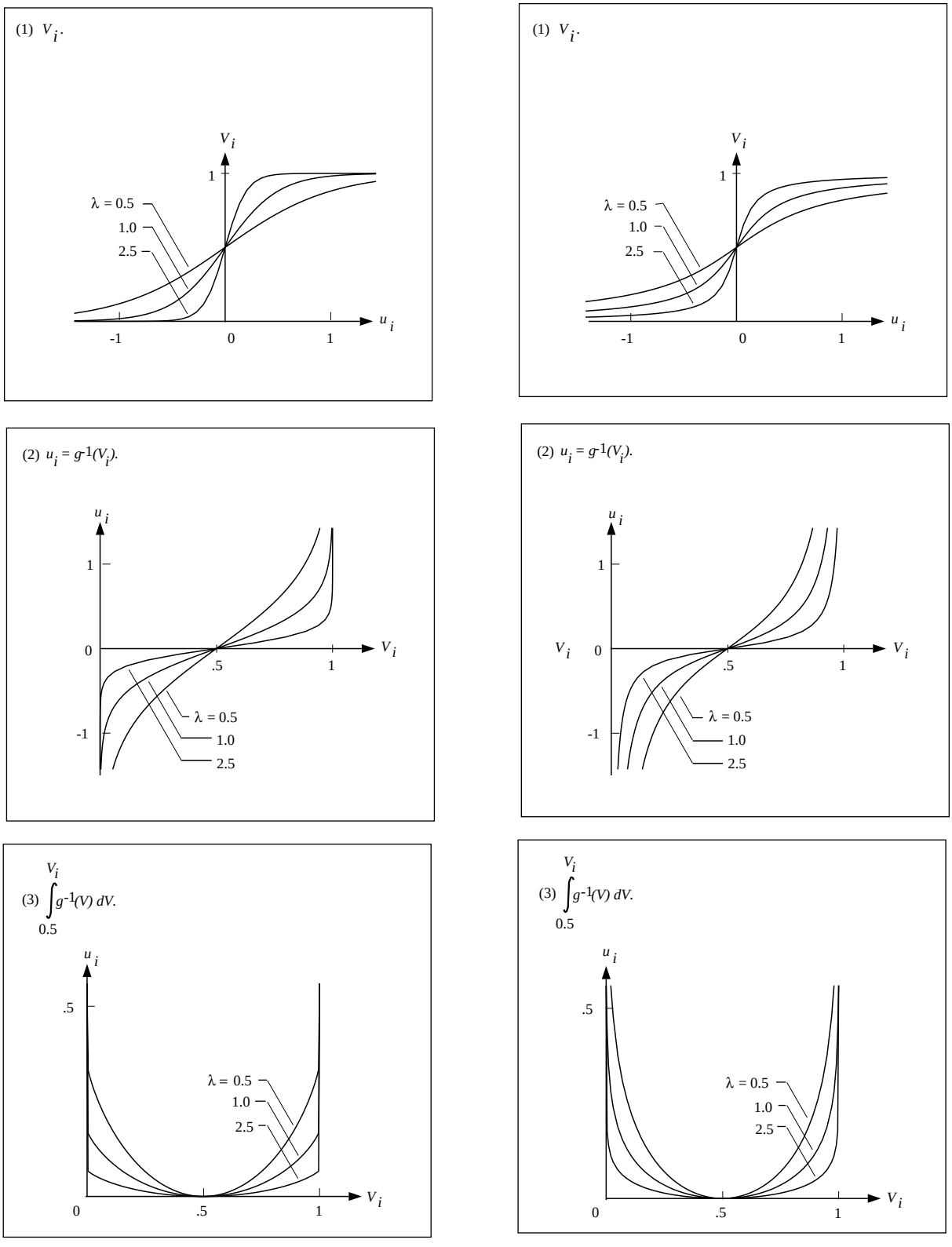
$$\frac{dE}{dt} = -\sum_i C_i \frac{dV_i}{du_i}\left(\frac{du_i}{dt}\right)^2 \tag{B8}$$

Assuming that the transfer function is monotonically increasing ($dV_i/du_i > 0$) and with $C_i > 0$, each term in the sum of equation (B8) is nonnegative. Thus,

$$\frac{dE}{dt} \leq 0 \tag{B9a}$$

and

$$\frac{dE}{dt} = 0 \tag{B9b}$$



(a) $V_i = \frac{1}{2}[1 + \tanh(2\lambda u_i)]$.

(b) $V_i = \frac{1}{2} + \frac{1}{\pi}[\arctan(\lambda \pi u_i)]$.

Figure B 1. Illustration of two different sigmoidal transfer functions $V_i = g(u_i)$, their inverse functions $u_i = g^{-1}(V_i)$, and their integrals for different values of gain λ.

(a) λ = 5.

(b) λ = 10.

(c) λ = 25.

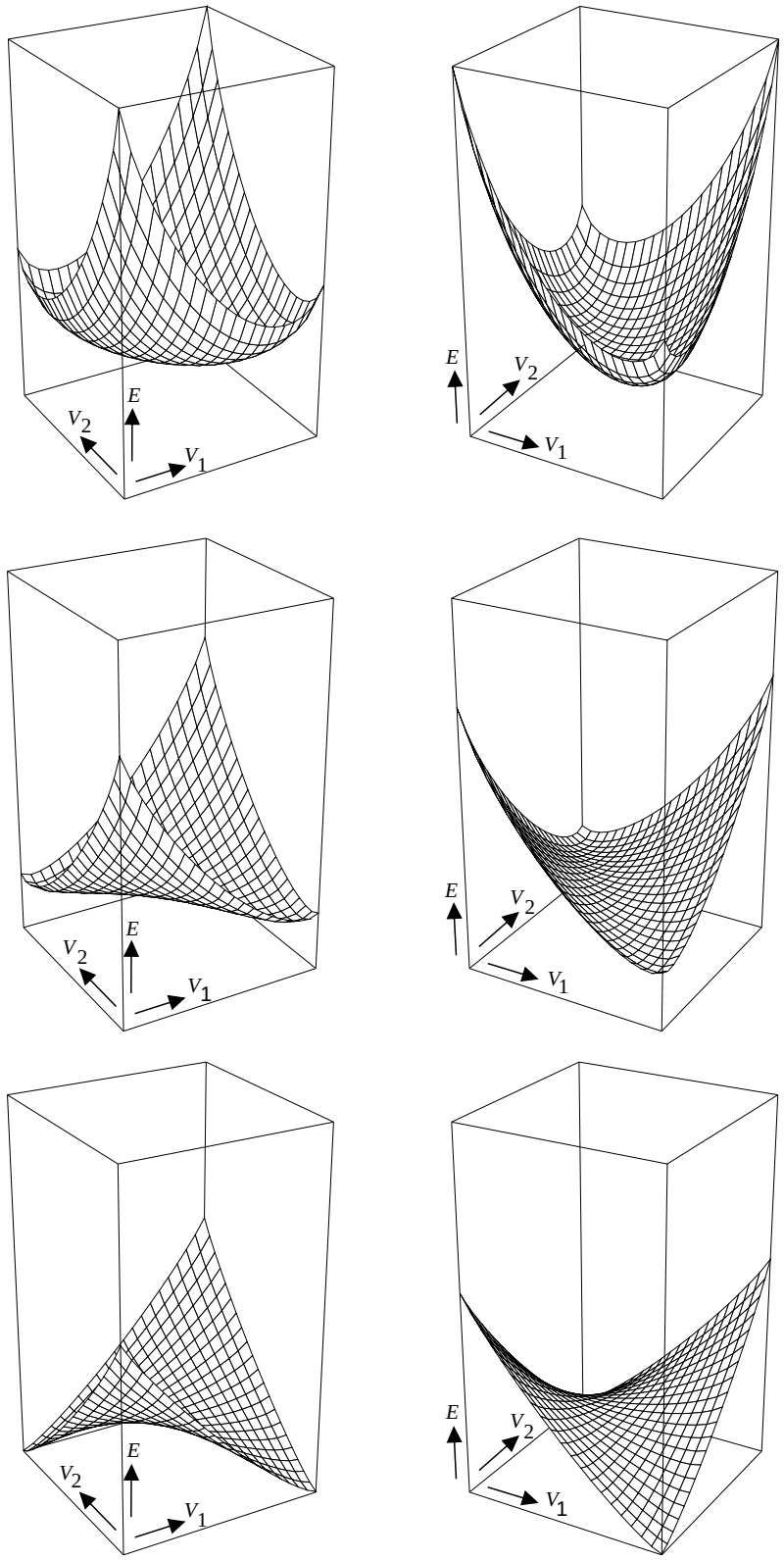


Figure B2. Plot of energy function according to equation (B1) for "Flip-Flop" in sketch A. $T_{ij} = -2$; $I_i = 1$; $r_i = 1$; $C_i = 1$; transfer function as shown in figure B1(a1).

when $du_i/dt = 0$ for all units i. Since the energy equation (B1) is bounded, equation (B9) implies that equation (B1) is a Liapunov function for the system equation (B6). This means that any time evolution of the system decreases the energy equation (B1) by moving to a local energy minimum at which point the motion of the system stops.

10. References

Anderson, James A. 1983: Cognitive and Psychological Computation With Neural Models. *IEEE Trans. Syst., Man, & Cybern.,* vol. SMC-13, no. 5. Sept./Oct., pp. 799–815.

Bannister, Joseph A.; and Trivedi, Kishor S. 1988: Task Allocation in Fault-Tolerant Distributed Systems. *Tutorial—Hard Real-Time Systems,* John A. Stankovic and Krithi Ramamritham, eds., IEEE Catalog No. EH0276-6, Computer Soc. of IEEE, pp. 256–272.

Belfore, Lee A., II; and Johnson, Barry W. 1989: The Fault-Tolerance of Neural Networks. *Neural Netw.,* vol. 1, no. 1, Jan., pp. 24–41.

Brandt, Robert D.; Wang, Yao; Laub, Alan J.; and Mitra, Sanjit K. 1988: Alternative Networks for Solving the Traveling Salesman Problem and the List-Matching Problem. *IEEE International Conference on Neural Networks,* IEEE Catalog No. 88CH2632-8, Inst. of Electrical and Electronics Engineers, Inc., pp. II-333–II-340.

Cohen, Michael A.; and Grossberg, Stephen 1983: Absolute Stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Networks. *IEEE Trans. Syst. Man, & Cybern.,* vol. SMC-13, no. 5, Sept./Oct., pp. 815–826.

Garey, Michael R.; and Johnson, David S. 1979: *Computers and Intractability—A Guide to the Theory of NP-Completeness.* W. H. Freeman and Co.

Grossberg, Stephen 1988: Nonlinear Neural Networks: Principles, Mechanisms, and Architectures. *Neural Netw.,* vol. 1, no. 1, pp. 17–61.

Hedge, Shailesh U.; Sweet, Jeffrey L.; and Levy, William B. 1988: Determination of Parameters in a Hopfield/Tank Computational Network. *IEEE International Conference on Neural Networks,* IEEE Catalog No. 88CH2632-8, Inst. of Electrical and Electronics Engineers, Inc., pp. II-291–II-298.

Hinton, G. E.; and Sejnowski, T. J. 1986: Learning and Relearning in Boltzmann Machines. *Parallel Distributed Processing—Explorations in the Microstructure of Cognition, Volume 1: Foundations,* David E. Rumelhart, James L. McClelland, and PDP Research Group, eds., MIT Press, pp. 282–317.

Hinton, Geoffrey E.; and Shallice, Tim 1989: *Lesioning a Connectionist Network: Investigations of Acquired Dyslexia.* TR-CRG-TR-89-3 (Grant 87-2-36), Dep. of Computer Science, Univ. of Toronto, May.

Hopfield, J. J. 1982: Neural Networks and Physical Systems With Emergent Collective Computational Abilities. *Proc. Natl. Acad. Sci. USA,* vol. 79, no. 8, Apr., pp. 2554–2558.

Hopfield, J. J. 1984: Neurons With Graded Response Have Collective Computational Properties Like Those of Two-State Neurons. *Proc. Natl. Acad. Sci. USA,* vol. 81, no. 10, May, pp. 3088–3092.

Hopfield, J. J.; and Tank, D. W. 1985: "Neural" Computation of Decisions in Optimization Problems. *Biol. Cybern.,* vol. 52, pp. 141–152.

Hutchinson, James M.; and Koch, Christof 1986: Simple Analog and Hybrid Networks for Surface Interpolation. *Neural Networks for Computing,* John S. Denker, ed., Volume 151 of *AIP Conference Proceedings,* American Inst. of Physics, pp. 235–240.

Kohonen, Teuvo 1988: *Self-Organization and Associative Memory,* Second ed. Springer-Verlag.

Lin, S.; and Kernighan, B. W. 1973: An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Oper. Res.,* vol. 21, pp. 498–516.

Marcus, C. M.; and Westervelt, R. M. 1989: Dynamics of Analog Neural Networks With Time Delay. *Advances in Neural Information Processing Systems,* Morgan Kauffman.

Page, Edward W.; and Tagliarini, Gene A. 1988: Algorithm Development for Neural Networks. Clemson University paper presented at the SPIE Symposium on Innovative Science and Technology (Los Angeles, California), Jan.

Palumbo, Daniel L.; and Butler, Ricky W. 1986: A Performance Evaluation of the Software-Implemented Fault-Tolerance Computer. *J. Guid., Control, & Dyn.,* vol. 9, no. 2, Mar.–Apr., pp. 175–180.

Pao, Yoh-Han 1989: *Adaptive Pattern Recognition and Neural Networks.* Addison-Wesley Publ. Co., Inc.

Petsche, Thomas; and Dickinson, Bradley W. 1990: Trellis Codes, Receptive Fields, and Fault Tolerant, Self-Repairing Neural Networks. *IEEE Trans. Neural Netw.,* vol. 1, no. 2, June, pp. 154–166.

Press, William H.; Flannery, Brian P.; Teukolsky, Saul A.; and Vetterling, William T. 1986: *Numerical Recipes—The Art of Scientific Computing.* Cambridge Univ. Press.

Protzel, Peter W. 1990: Comparative Performance Measure for Neural Networks Solving Optimization Problems. *International Joint Conference on Neural Networks, Volume II—Applications Track,* Lawrence Erlbaum Assoc., Inc., Publ., pp. II-523–II-526.

Protzel, P.; Palumbo, D.; and Arras, M. 1989: *Fault-Tolerance of a Neural Network Solving the Traveling Salesman Problem.* NASA CR-181798, ICASE Rep. No. 89-12.

Rumelhart, David E.; McClelland, James L.; and PDP Research Group, eds. 1986: *Parallel Distributed Processing—Explorations in the Microstructure of Cognition, Volume 1: Foundations.* MIT Press.

Sejnowski, Terrence J.; and Rosenberg, Charles R. 1986: *NETtalk: A Parallel Network That Learns To Read Aloud.* JHU/EECS-86/01, Johns Hopkins Univ.

Smith, Michael J. S.; and Portmann, Clemenz L. 1989: Practical Design and Analysis of a Simple "Neural" Optimization Circuit. *IEEE Trans. Circuits & Syst.,* vol. 36, no. 1, Jan., pp. 42–50.

Syslo, Maciej M.; Deo, Narsingh; and Kowalik, Janusz S. 1983: *Discrete Optimization Algorithms—With Pascal Programs.* Prentice-Hall, Inc.

Tagliarini, Gene A.; and Page, Edward W. 1987: A Neural-Network Solution to a Concentrator Assignment Problem. *1987 IEEE Conference on "Neural Information Processing Systems—Natural and Synthetic,"* IEEE Catalog No. 87CH2386-1, Inst. of Electrical and Electronic Engineers and American Physical Soc., p. 38.

Tank, David W.; and Hopfield, John J. 1986: Simple "Neural" Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit. *IEEE Trans. Circuits & Syst.,* vol. CAS-33, no. 5, May, pp. 533–541.

Van den Bout, David E.; and Miller, T. K. 1988: A Traveling Salesman Objective Function That Works. *IEEE International Conference on Neural Networks,* IEEE Catalog No. 88CH2632-8, Inst. of Electrical and Electronics Engineers, Inc., pp. II-299–II-303.

Wasserman, Philip D. 1989: *Neural Computing—Theory and Practice.* Van Nostrand Reinhold.

Wilson, G. V.; and Pawley, G. S. 1988: On the Stability of the Traveling Salesman Problem Algorithm of Hopfield and Tank. *Biol. Cybern.,* vol. 58, pp. 63–70.

Zornetzer, Steven F.; Davis, Joel L.; and Lau, Clifford, eds. 1990: *An Introduction to Neural and Electronic Networks.* Academic Press, Inc.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1992	Technical Paper

4. TITLE AND SUBTITLE
Fault Tolerance of Artificial Neural Networks With Applications in Critical Systems

5. FUNDING NUMBERS
WU 307-50-10-12

6. AUTHOR(S)
Peter W. Protzel, Daniel L. Palumbo, and Michael K. Arras

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23665-5225

8. PERFORMING ORGANIZATION REPORT NUMBER
L-16969

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA TP-3187

11. SUPPLEMENTARY NOTES
Protzel and Arras: ICASE, Hampton, VA; Palumbo: Langley Research Center, Hampton, VA.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified–Unlimited

Subject Category 62

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

One of the key benefits of future hardware implementations of certain artificial neural networks (ANN's) is their apparently "built-in" fault tolerance which makes them potential candidates for critical tasks with high reliability requirements. This paper investigates the fault-tolerance characteristics of time-continuous, recurrent ANN's that can be used to solve optimization problems. The principle of operation and the performance of these networks are first illustrated by using well-known model problems like the traveling salesman problem and the assignment problem. The ANN's are then subjected to up to 13 simultaneous "stuck-at-1" or "stuck-at-0" faults for network sizes of up to 900 "neurons." The effect of these faults on the performance is demonstrated and the cause for the observed fault tolerance is discussed. An application is presented in which a network performs a critical task for a real-time distributed processing system by generating new task allocations during the reconfiguration of the system. The performance degradation of the ANN under the presence of faults is investigated by large-scale simulations, and the potential benefits of delegating a critical task to a fault-tolerant network are discussed.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Artificial neural networks; Fault tolerance; Performance measure	49
	16. PRICE CODE: A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		